

Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Bot na platformě Discord



2023

Vedoucí práce:
Mgr. Roman Vyjídaček

Lukáš Netřeba

Studijní program: Informatika, prezenční
forma

Bibliografické údaje

Autor: Lukáš Netřeba
Název práce: Bot na platformě Discord
Typ práce: bakalářská práce
Pracoviště: Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby: 2023
Studijní program: Informatika, prezenční forma
Vedoucí práce: Mgr. Roman Vyjídáček
Počet stran: 53
Přílohy: elektronická data v úložišti katedry informatiky
Jazyk práce: český

Bibliographic info

Author: Lukáš Netřeba
Title: Bot on Discord platform
Thesis type: bachelor thesis
Department: Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense: 2023
Study program: Computer Science, full-time form
Supervisor: Mgr. Roman Vyjídáček
Page count: 53
Supplements: electronic data in the storage of department of computer science
Thesis language: Czech

Anotace

Obsahem práce je vytvoření Bota pro uživatele a komunity nacházející se na komunikační platformě Discord. Tato aplikace umožní bota používat pro správu komunit, nastavení jejich pravidel, pravomocí a moderace chatu. Bot umožní vytvářet a organizovat události, na které se uživatelé z komunit budou moci přihlásit a být upozorněni před jejich začátkem. Bude moci zprostředkovávat audio přehrávač skrze hlasové kanály na serverech.

Synopsis

Content of this thesis aims to create a Bot for users and communities on Discord platform. Application will provide management for those communities with their own set of rules, permissions and chat moderation. The bot will take care of creating and organizing scheduled events for community users that can participate in and be notified just before the event starts. The bot will also mediate an audio player in voice channels on the servers.

Klíčová slova: Discord, nástroj pro správu, Python, Bot, audio přehrávač, události

Keywords: Discord, administration tool, Python, Bot, audio player, scheduled events

Chtěl bych poděkovat svému vedoucímu práce, panu Mgr. Romanu Vyjídáčkovi, za to, že mi umožnil zpracovat toto téma a za jeho odbornou pomoc při jeho zpracování. Dále bych chtěl poděkovat svým blízkým a kolegům, kteří byli se mnou trpěliví a motivovali mě k vypracování práce.

Odevzdáním tohoto textu jeho autor/ka místopřísežně prohlašuje, že celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

Obsah

| | | |
|----------|--|-----------|
| 1 | Úvod | 9 |
| 2 | Platforma Discord | 10 |
| 2.1 | Definice a terminologie | 10 |
| 2.2 | Servery a komunity | 11 |
| 2.3 | Kanály hlasové a textové | 12 |
| 2.4 | Uživatelé, role a pravomoce | 12 |
| 2.5 | Limitace a předplatné | 13 |
| 3 | Discord Bot | 14 |
| 3.1 | Ovládání a práce s botem | 15 |
| 3.1.1 | Prefixové zprávy | 16 |
| 3.1.2 | Lomítkové příkazy | 16 |
| 3.2 | Technologie pro vývoj | 17 |
| 3.2.1 | Discord API | 17 |
| 3.2.1.1 | WebSocket API | 18 |
| 3.2.1.2 | REST API | 18 |
| 3.2.2 | Asyncio | 19 |
| 3.2.3 | FFmpeg | 20 |
| 3.3 | Knihovny pro práci s Discord API | 20 |
| 3.3.1 | Knihovna discord.py | 21 |
| 3.4 | Možnosti hostování aplikace | 21 |
| 4 | Návrh funkcionalit | 22 |
| 4.1 | Automatické vlastnosti | 22 |
| 4.1.1 | Událost: nový člen | 22 |
| 4.1.1.1 | Zaslání pravidel | 23 |
| 4.1.1.2 | Přiřazení role, pravomocí | 23 |
| 4.1.1.3 | Kontrola záznamů | 23 |
| 4.1.2 | Událost: zpráva | 24 |
| 4.1.2.1 | Kontrola odeslaných a upravených zpráv | 24 |
| 4.2 | Manuální vlastnosti | 25 |
| 4.2.1 | Správa uživatelů | 25 |
| 4.2.2 | Správa událostí | 25 |
| 4.2.3 | Přehrávání audio stop | 26 |
| 5 | Implementace | 27 |
| 5.1 | Souborová struktura | 27 |
| 5.2 | Spouštěcí soubor | 28 |
| 5.3 | Vytvoření aplikace a účtu jako bot | 30 |
| 5.4 | Ukládání dat | 32 |
| 5.5 | Struktura příkazů v kódu | 34 |
| 5.6 | Pravidla komunit | 36 |

| | | |
|------|---|-----------|
| 5.7 | Kontrola zpráv | 39 |
| 5.8 | Kontrola uživatelů, přidělení pravomocí | 41 |
| 5.9 | Události a jejich spravování | 43 |
| 5.10 | Přehrávač | 46 |
| 5.11 | Hosting aplikace bota | 48 |
| | Závěr | 49 |
| | Conclusions | 50 |
| | A Obsah přiloženého datového média | 51 |
| | Seznam zkratk | 52 |
| | Literatura | 53 |

Seznam obrázků

| | | |
|----|--|----|
| 1 | Systémový bot platformy Discord | 15 |
| 2 | Příkaz vyvolaný prefixovanou zprávou | 16 |
| 3 | Kontextové menu lomítkových příkazů | 17 |
| 4 | Způsob komunikace bota a platformy [7] | 18 |
| 5 | Asynchronní programování s jedním procesem [13] | 19 |
| 6 | Obslužení události při příchodu nového uživatele | 23 |
| 7 | Obslužení události nové (upravené) zprávy | 24 |
| 8 | Souborový systém aplikace bota | 27 |
| 9 | Vytvoření aplikace | 31 |
| 10 | Vytvoření aplikace jako bot | 31 |
| 11 | Získání autorizačního tokenu bota | 32 |
| 12 | Identifikátor aplikace | 32 |
| 13 | Ukázka zaslaných pravidel novým členům | 36 |
| 14 | Ukázka nastavení automatické role | 42 |
| 15 | Ukázka automatického přidělení role | 43 |
| 16 | Ukázka vytvořené události příkazem <i>events echat</i> | 44 |
| 17 | Ukázka zaregistrování audia | 46 |
| 18 | Ukázka přehrávání audia botem | 47 |

Seznam tabulek

| | | |
|---|--|----|
| 1 | Základní limity | 13 |
| 2 | Úrovně limitů pro předplatitelské účty | 14 |
| 3 | Úrovně limitů pro předplacené servery | 14 |
| 4 | Knihovny pro práci s boty [12] | 21 |

Seznam zdrojových kódů

| | | |
|----|--|----|
| 1 | Nastavení oprávnění bota v Discord API | 28 |
| 2 | Funkce určující prefix | 29 |
| 3 | Vytvoření bota | 29 |
| 4 | Ukázka callback funkce websocket eventu | 29 |
| 5 | Synchronizace příkazů | 30 |
| 6 | Instalace modulů | 30 |
| 7 | Spuštění bota | 30 |
| 8 | Otevření databázového souboru | 33 |
| 9 | Zápis do databázového souboru | 33 |
| 10 | Čtení v databázovém souboru | 33 |
| 11 | Aktualizace dat v databázovém souboru | 34 |
| 12 | Ukázka struktury příkazu v kódu, příkaz <i>clear</i> | 35 |
| 13 | Chybová funkce k příkazu <i>clear</i> | 36 |

| | | |
|----|---|----|
| 14 | Event (websocket) připojený člen | 37 |
| 15 | Zaslání pravidel komunity | 37 |
| 16 | Zobrazení pravidel | 38 |
| 17 | Přidání pravidla | 38 |
| 18 | Resetování pravidel | 39 |
| 19 | Smazání pravidla | 39 |
| 20 | Event (websocket) nové zprávy | 40 |
| 21 | Kontrola obsahu zprávy | 41 |
| 22 | Event (websocket) upravené zprávy | 41 |
| 23 | Zjištění existence záznamu uživatele | 42 |
| 24 | Oznámení o existenci záznamů nového uživatele | 42 |
| 25 | Část funkce pro vytvoření události | 45 |
| 26 | Část funkce o oznámení události | 46 |
| 27 | Přehrání audia příkazem <i>play</i> | 47 |
| 28 | Zprocesování a přehrání audia | 48 |

1 Úvod

Tato práce se zabývá vytvořením bota pomocí knihovny discord.py v programovacím jazyce Python pro komunikační platformu Discord, který bude sloužit jako nástroj pro komunity ke zlepšení správy nad danou komunitou, jejími pravidly, nastavením pravomocí pro nově připojené uživatele a jejich následnou kontrolou vůči prohřeškům z jiných komunit, kde bot mohl problémového uživatele již spatřit a případně tak závčas varovat moderátory. Bot umožní vytvoření události s názvem, popiskem a datem konání a evidencí přihlášených (odmítnutých, nerozhodných) uživatelů na tuto událost a těsně před započítím události je upozorní. Dále zprostředkuje možnost posílat audio stopu z YouTube přímo do hlasových kanálů na serveru. V práci jsou rozebrány způsoby, kde lze takovou aplikaci provozovat, vlastnosti každé z možností a realizace jedné z nich.

Teoretická část textu seznamuje s platformou Discord po její uživatelské a technické stránce, které jsou nezbytné k pochopení a vypracování práce. Z uživatelské stránky je zde popsána používaná terminologie v kontextu s touto platformou a také vlastnosti této telekomunikační platformy. Z technického pohledu je rozebráno, jaké technologie platforma Discord využívá ke své činnosti. Jakým způsobem tyto technologie fungují a jak se dají využít v rámci vytváření vlastních aplikací.

Praktická část se zabývá efektivním návrhem a následnou implementací bota. Během návrhu je každá funkcionalita podrobně popsána, jakým způsobem by s ní měl uživatel interagovat, případně popis její cílené činnosti. Implementace obsahuje vhodnou realizaci navržených funkcionalit bota včetně jejich demonstrace.

Výsledkem práce je funkční aplikace realizující bota, který může být nasazen pro jakoukoliv komunitu na Discordu.

2 Platforma Discord

Discord je VoIP sociální komunikační platforma. Její uživatelé zde mají možnost komunikovat skrze audiohovor, videohovor, média, soubory a zprávy, a to buď prostřednictvím soukromých chatů a nebo jako součástí komunit (veřejných, soukromých), kterým se také někdy říká „servery“ nebo „guildy“. Discord je multiplatformní aplikace a běží na operačních systémech Windows, macOS, Linux, Android, iOS či ve webových prohlížečích. V roce 2021 bylo na Discordu registrováno 350 milionů uživatelů a evidováno 150 milionů aktivních uživatelů denně.

Platforma vznikla jako nápad dvou studentů pro odvětví herního průmyslu, kteří cítili nedostatky v komunikaci již existujících VoIP software se spoluhráči v taktických hrách jako je Final Fantasy XIV. [1] První release Discordu tak byl začátkem roku 2015. Dnes po letech vývoje se používá nejen v odvětví herního průmyslu ačkoliv ten stále dominuje. [2]

Discord je řazen mezi podobné platformy jako jsou např. Microsoft Teams, Microsoft Skype, Slack, Teamspeak, Reddit, Facebook Groups, které všechny patří ke komunikačním platformám s různými rozdíly ve funkcionalitě podle uživatelů na které jsou cíleny.

2.1 Definice a terminologie

Discord používá svoji vlastní terminologii, kterou respektuje i tento text a je používán ve zbytku této práce. Následující seznam vysvětlí nejpodstatnější termíny:

Server

Server neboli komunitní server, někdy také označován anglicky slovem *guild* v kontextu kódu, je v uživatelském rozhraní platformy uskupení sdružující uživatele, hlasové a textové kanály.

Kanál

Kanál je místo, skrze které mohou uživatelé serveru komunikovat podle jeho typu. Existují pouze dva typy kanálů a to *hlasové* nebo *textové*. Textové přenášejí zprávy, média a soubory, zatímco hlasové přenášejí navíc audio a video.

Kategorie

Kategorie je pojmenované uskupení kanálů, které jsou v uživatelském rozhraní platformy zobrazeny jako složky. Kategorie mohou obsahovat další kategorie a kanály.

Role

Role je pojmenovaná skupina uživatelů, které mohou být v uživatelském rozhraní platformy zobrazeny jako složky. Každá role s sebou nese svoji množinu oprávnění a barvu k zobrazení. Role jsou postaveny v lineární hierarchii, role tak bývá nadřazena/podřazena jiné roli.

Emoji

Discord emoji je spojením *Unicode* emoji a rozšířením o personalizované obrázky definované v rámci konkrétního discord serveru.

Reakce

Reakce je discord emoji, které pomocí interakcí s tlačítkem u libovolné zprávy mohl uživatel nebo bot přidat.

Uživatel

Uživatel je osoba, která má vytvořený účet na platformě a může se tak připojit k libovolnému serveru. Uživatelé se dělí na dva druhy: *člověk* nebo *bot*. Když je řečeno slovo uživatel, je tím myšlen druh účtu, který není automatizovaný a stojí za ním skutečný člověk.

Bot

Bot je speciální druh uživatele, jehož účet je řízený programem, který komunikuje s *Discord API* [4]. Má přístup k více funkcím než obyčejný uživatel, které může používat k různým úkonům. Navíc bývá méně limitován než běžný uživatel v počtu zasílaných požadavků do API.

Příkaz

Příkaz je textová zpráva napsaná v uživatelském rozhraní v poli pro odesílání zpráv. Příkazy se dělí na dva typy, *prefixové* nebo *lomítkové*. Prefixové mají serverem určený speciální symbol, který rozlišuje normální zprávu od příkazové k vyhodnocení botem. Tato zpráva je odeslána do chatu jako běžná zpráva. Lomítkový příkaz se neodesílá do chatu jako prefixový příkaz, ale je odchycen v *Discord API* a předaný botu k obslužení.

2.2 Servery a komunity

Server na platformě Discord je místo, kde se mohou shlukovat uživatelé na jednom místě a komunikovat přes kanály ať hovorem či skrze text. Server je v uživatelském rozhraní jako kulatá ikonka v jeho levé části klientské aplikace či ve webovém prohlížeči. Není zde myšleno, že server je výkonný počítač, který obsluhuje požadavky. Discord server, synonymem komunita či guilda, je poskytován a hostován přímo službou discordu jako takovou, která však běží na fyzických strojích společnosti, které obsluhují několik těchto instancí komunit zároveň.

Na server se může uživatel dostat pouze pomocí pozvánky (vloženou v **UI**), která může být krátkým textovým řetězcem připomínajícím hash sloužící jako

identifikátor pozvánky. Popřípadě [URL](#) adresa, kde je zmíněný identifikátor obsažen v jejím parametru. Pozvánka může být implicitně parametrizována navíc o dobu expirace popřípadě limitem počtu užití. Neomezené pozvánky se zpravidla používají pro veřejné servery.

Při vytváření vlastního discord serveru lze v průvodci nastavením najít i šablony pro různé účely, které pomohou vytvořit a základně nastavit discord server podle specifických potřeb komunity.

Na serveru se vyskytují kategorie, textové a hlasové kanály a samotní uživatelé serveru a případné integrace aplikací, které mají programem řízený účet odlišený za jménem pomocí štítku s nápisem „BOT“.

2.3 Kanály hlasové a textové

Kanál je prostředník komunikace na platformě, dělí se zpravidla na hlasové a textové, kde soukromý chat je speciální druh kanálu pouze mezi dvěma uživateli. Kanály obecně mají společnou vlastnost, která podléhá oprávnění podle rolí nebo také „per user“ a nastavením funkčnosti.

Textové kanály mohou být dle oprávnění pro uživatele neviditelné, zamčené pro psaní, zakázané v jiných směrech např. posílání odkazů, přidávání reakcí, možnost smazat vlastní nebo cizí zprávu, nebo nastavené na určitý časový limit pro odesílání zpráv.

Hlasové kanály opět podle oprávnění mohou být neviditelné, zaheslované nebo nastavené na maximální limit současně připojených uživatelů popřípadě nastavené v kvalitě přenosu audia a videa.

Speciálním případem je komunikace uživatel s jiným uživatelem mimo server přes soukromý chat nebo hovor. Zde neplatí žádné oprávnění a uživatelé nejsou limitováni oprávněními, které lze aplikovat na role. Jedinou výjimkou jsou limity nastavené samotnou platformou, mezi které patří znemožnění smazání zpráv druhého uživatele, volitelné znemožnění komunikace s druhým uživatelem mimo seznam jeho přátel a sdílený společného serveru. Posledním limitem je zde zamezení komunikace, kdy příjemce zablokoval odesílatele.

2.4 Uživatelé, role a pravomoce

Uživatel je kdokoliv kdo provedl registraci na platformě Discord a zavazuje se k dodržení podmínek služby, v rámci serveru může mít uživatel jednu či více rolí, které určují jeho pravomoce.

Role je skupina oprávnění vztahující se pouze na serveru na němž byla vytvořena a může být přiřazena pouze uživatelům tohoto serveru.

K základní roli, kterou má každý server je role *everyone* s výchozím nastavením oprávnění, které se zpravidla ponechává a vytvářejí se role nové. Pro zpřísnění oprávnění než je role *everyone* bývá zvykem vytvořit novou roli, kterou některý z botů na serveru nastaví nově přichozím. Tato základní role však slouží v chatové zprávě pro hromadné označení všech členů serveru pomocí *@everyone*.

Existuje jedna role implicitní, kterou je *server owner*, dá rozpoznat v seznamu uživatelů serveru má uživatel za svým jménem ikonku královské koruny. Role není zobrazena v seznamu rolí serveru a není možné s ní nijak zacházet, má nejvyšší administrátorské oprávnění nade všemi.

Role na discord serverech jsou vytvářeny jako pořadový seznam v nastavení serveru, toto pořadí určuje jejich lineární hierarchii. Což znamená, že role nadřazená může dělat úkony na roli podřazené.

Pravomoce jsou elementární úkony, které mohou být nastaveny konkrétní roli. Mezi takové úkony např. patří: právo smazat zprávu, vytvořit pozvánku, vyhodit uživatele, upravit název kanálu, upravit nastavení serveru, vytvořit nové role. Všechny úkony lze vidět v uživatelském rozhraní serveru pro správu rolí.

2.5 Limitace a předplatné

Myšleny jsou zde limity, které jsou kladeny na uživatele a komunitní servery na platformě s možností jejich výběru na základě předplatného. Limitem aplikací pak rozumíme omezení, které jsou kladeny na účet bota a jeho přístup k informacím platformy.

Některé limity je schopen si uživatel nebo komunitní server navýšit díky předplatitelům. Pro uživatele se druh předplatného nazývá *Nitro Basic* a vylepšená varianta *Nitro*. Předplacený server se nazývá dle její úrovně, které jsou *Tier 1*, *Tier 2* a *Tier 3*.

Základní limity, podle tabulky 1, jsou platné pro všechny uživatele a komunitní servery. Většina, až na výjimky (*), nelze ovlivnit zakoupením předplatného pro server nebo uživatelský účet.

| Typ | Limit |
|--|---------|
| Počet uživatelů na serveru | 250,000 |
| Počet online uživatelů na serveru | 5,000 |
| Počet serverů uživatel smí být členem* | 100 |
| Počet kategorií na serveru | 50 |
| Počet kanálů na serveru | 500 |
| Počet rolí na serveru | 250 |
| Počet neanimovaných emoji na serveru | 50 |
| Počet animovaných emoji na serveru | 50 |

Tabulka 1: Základní limity

Limity s předplatným se v případě předplatného pro uživatele řadí do dvou úrovní, pro komunitu se předplatné řadí do úrovní tří. Tabulka 2 ukazuje úrovně předplatného pro uživatele, tabulka 3 ukazuje předplatné úrovně pro servery.

K limitům patří také „rate limit“, což je maximální počet požadavků, které uživatelský účet nebo účet bota může platformě zasílat, bot má tento limit vyšší. Požadavek je např. odeslání zprávy, vytvoření pozvánky serveru nebo probíhající

| Typ | Úroveň 1 | Úroveň 2 |
|---------------------------------------|---------------|---------------|
| Vlastní emoji kdekoliv | Ano | Ano |
| Vlastní samolepky kdekoliv | Ano | Ano |
| Velikost souborů | 50MB | 500MB |
| Vysoké rozlišení videa | Ne | Ano, až 4k@60 |
| Personalizovaný profil účtu | Ne | Ano |
| Personalizovaný profil „per server“ | Ne | Ano |
| Odznak předplatitele | Ano | Ano |
| Vlastní pozadí ve videohovorů | Ano | Ano |
| Počet serverů uživatel smí být členem | 100 | 200 |
| Delší zprávy | do 2000 znaků | do 4000 znaků |

Tabulka 2: Úrovně limitů pro předplatitelské účty

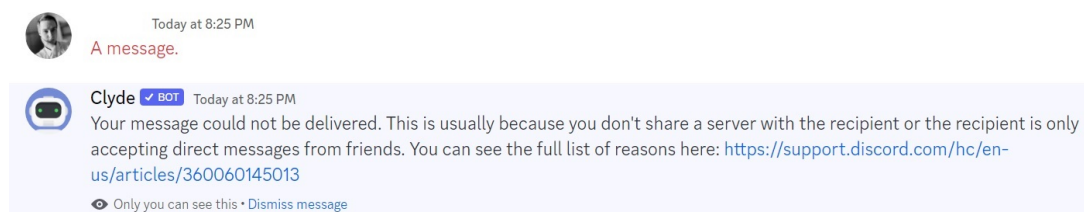
| Typ | Úroveň 1 | Úroveň 2 | Úroveň 3 |
|----------------------------------|------------|------------|------------|
| Vlastních emoji serveru | až 100 | až 150 | až 250 |
| Audio kvalita | až 128Kbps | až 256Kbps | až 384Kbps |
| Vlastní pozadí v pozvánce | Ano | Ano | Ano |
| Vlastních samolepek serveru | až 15 | až 30 | až 60 |
| Animovaná ikona serveru | Ano | Ano | Ano |
| Kvalita 1080p@30 videa všem | Ano | Ano | Ano |
| Vlastní ikony rolí | Ne | Ano | Ano |
| Vlastní banner serveru | Ne | Ano | Ano |
| Kvalita 1080@60 videa všem | Ne | Ano | Ano |
| Až 50MB velikost souborů všem | Ne | Ano | Ano |
| Až 100MB velikost souborů všem | Ne | Ne | Ano |
| Animovaný vlastní banner serveru | Ne | Ne | Ano |
| Vlastní odkaz pozvánky | Ne | Ne | Ano |

Tabulka 3: Úrovně limitů pro předplacené servery

audio či videohovor. Při rychlém vyčerpání přiděleného maximálního počtu požadavků pro určitý časový úsek požadavky nemají žádný efekt, dokud neuplyne určitý čas, po kterém se předešlé požadavky automaticky pokusí sami zopakovat. Nadměrným a úmyslným čerpáním tohoto maximálního počtu požadavků je proti [ToS](#) a může dojít k pozastavení nebo odstranění účtu jedince nebo komunitního serveru.

3 Discord Bot

Discord bot je uživatelský účet, který je kompletně automatizovaný programem, tento program pak komunikuje prostřednictvím Discord API [5] s platformou, na které provádí automatické úkony nebo úkony vyvolané uživatelem či prostředím



Obrázek 1: Systémový bot platformy Discord

za účelem vylepšení uživatelské zkušenosti. Platforma komunikuje s botem skrze WebSocket API [6] a předává tak v „realtime“ informace bez nutnosti navazovat neustále nová spojení. Bot si tyto odpovědi ukládá do mezipaměti, jedná se tak o rychlý přenos informace mezi platformou a botem. Druhým směrem bot odpovídá platformě skrze její REST API [7], kde místo odesílání celých objektů odesílá pouze identifikátory objektů, které má ve své mezipaměti.

Bota si může zřídit jakýkoliv registrovaný uživatel discordu, což může učinit na oficiálních webových stránkách platformy, kde svoji aplikaci pojmenuje a vytvoří. Vytvořením dostane autorizační token, který poté následně použije ve zdrojovém kódu podle patřičné implementace Discord API knihovny pro zvolený programovací jazyk.

Clyde bot je bot, jenž nepatří žádnému registrovanému uživateli, ale jedná se o systémového bota platformy, který je zde pro vylepšení a používání této platformy. Můžeme ho spatřit na obr. 1, když se pokusíme poslat zprávu uživateli, který nás zablokoval.

3.1 Ovládání a práce s botem

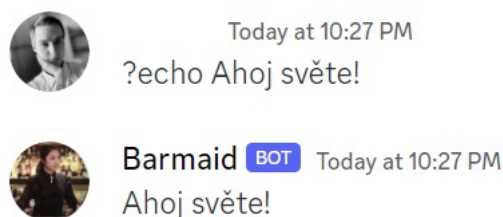
Ovládání bota je buď automatické, kdy bot řídí do určité míry svoje ovládání sám, nebo manuální, kdy vyžaduje vstup od uživatele. Automatické fungují tak, že přes websocket dostává bot informaci o eventech, které se na platformě stali, třeba uživatel odeslal zprávu a botu přijde event typu *on-message*, pro který ve zdrojovém kódu bude vytvořena funkce, která funguje jako *callback* a bude spuštěna při tomto eventu z websocketu. Tyto eventy jsou předdefinované a je na programátorovi, které funkce jako callbacky vytvoří podle toho, kterým eventům chce jeho aplikace naslouchat. Manuální jsou funkce, které definují příkaz prostřednictvím uživatelského rozhraní v chatovacím okně dvěma způsoby: prefixovanou zprávou nebo příkaz za lomítkem.

Pokud bot obsahuje funkce, které jsou v příslušném jazyce knihovny Discord API označeny jako event a dodržují definované pojmenování, pak budou automaticky skrze knihovnu spuštěny jako callback.

Uživatel bota nemusí v případě úkonů, které reagují na websocketem zaslané eventy, nic dělat. Pro úkony, které vyžadují zásah nebo doplnění informací k jejich provedení, musí uživatel tyto úkony vyvolat manuálně sám přes příkazy prefixové nebo lomítkové.

3.1.1 Prefixové zprávy

Prefixové zprávy jsou způsobem komunikace mezi uživatelem a botem, kdy uživatel napíše zprávu do chatu jako obvykle s rozdílem, že před začátek zprávy dá speciální symbol, který bot poté rozpozná, která zpráva obsahuje příkaz a která zpráva byla mezi běžnou probíhající konverzací na komunitním serveru. Mezi speciální symboly, které se často používají jsou: `?`, `!`, `&`, `*`. Tyto symboly jsou vždy na začátku zprávy a následuje za nimi řetězec znaků, který je definován jako příkaz. Příkaz je pak rozdělen na jeden či více argumentů pomocí mezer, před první mezerou je tak symbolické jméno příkazu.



Obrázek 2: Příkaz vyvolaný prefixovanou zprávou

Na obr. 2 můžeme vidět příkaz `echo` s jedním argumentem následujícím za první mezerou po slově `echo`. Při implementaci funkce příkazu je vždy možnost pro poslední argument příkazu posbírat celý řetězec včetně mezer a chápat jej jako jeden argument, jako tomu je v tomto případě.

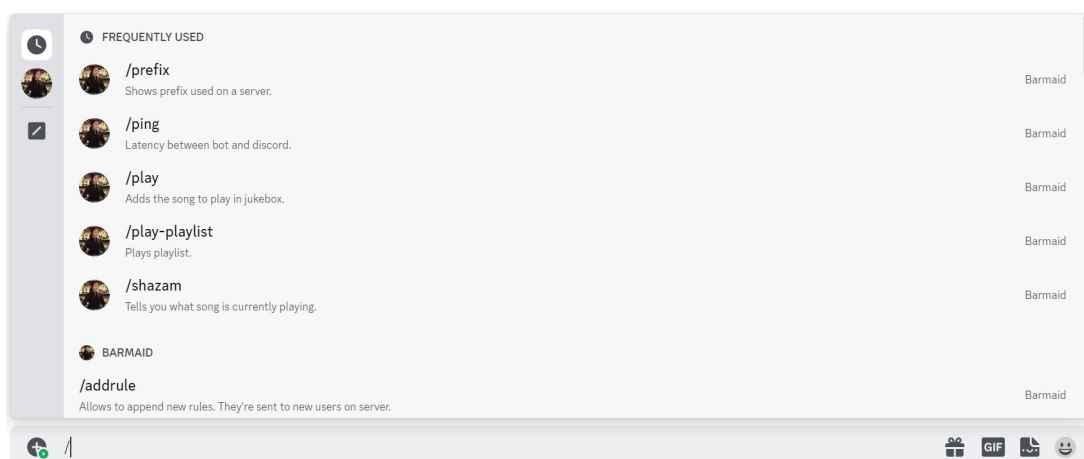
Pokud je příkaz s více argumenty, kdy do jednoho argumentu je potřeba dát řetězec obsahující i mezery, je potřeba tento řetězec obalit do uvozovek.

3.1.2 Lomítkové příkazy

Lomítkový příkaz je novější způsob komunikace uživatele s botem, kdy uživatel předepíše do textového pole pro odeslání nové zprávy symbol lomítka a zobrazí se mu kontextové menu s výběrem všech dostupných příkazů.

V uživatelském rozhraní se rozbalí menu s možnostmi příkazů, které jsou definovány v kódu bota. Uživatel vybere příkaz a je mu našeptáváno jaké hodnoty argumentů je nutné zadat pro úspěšné vykonání příkazu. Poprvé byly lomítkové příkazy přidány do uživatelského prostředí a do odpovídajících API koncem roku 2020, v roce 2022 se stali povinností pro boty, kteří musejí být verifikováni. Verifikovaný bot je zkontrolován vůči práci s citlivými daty uživatelů platformy a jeho následnému nakládání s nimi. Bot, který nepřekročí počet 100 komunitních serverů, ke kterým se připojil, nepotřebuje ověření. O toto ověření lze požádat až při překročení hranice.

Na obr. 3 vlevo vidíme sloupec, ve kterém nalezneme všechny boty na daném komunitním serveru. Uprostřed nabídku se všemi příkazy, kde shora jako první se zobrazují nejčastěji používané příkazy a na konci řádku příkazu vidíme název



Obrázek 3: Kontextové menu lomítkových příkazů

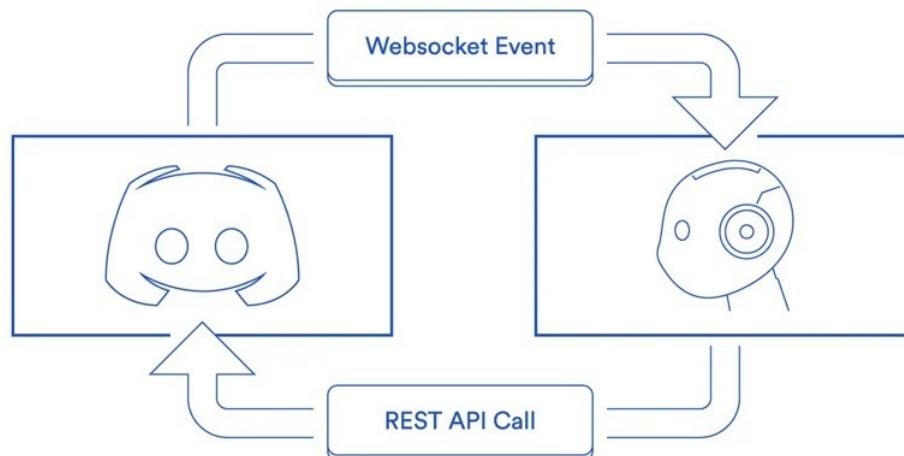
bota, kterému příkaz patří. Více botů může mít stejně pojmenovaný příkaz a uživatel je schopen je rozlišit podle profilové fotky nebo jména bota.

3.2 Technologie pro vývoj

Pro komunikaci s Discord platformou má sama platforma své vlastní [API](#), k počátku roku 2023 se datuje už 10. verze. Dá se používat ještě verze předešlá, verze 6-8 jsou nyní zastaralé a verze 1-5 jsou již zrušené. [5] Komunikovat s tímto API napřímo je náchylné na chyby programátora při implementaci, které mohou způsobit nechtěné přetěžování API, které může vyústit k zablokování této komunikace platformou nebo dokonce účtu bota či samotného uživatele vlastního tohoto bota. Od těchto strastí s verzí a správnou komunikací nás abstrahují knihovny pro práci s API platformy Discord. Knihovny jsou napsány jako „wrapper“ v patřičném programovacím jazyce usnadňující práci s API a ošetřují jeho nebezpečného zacházení. Další knihovny pro vybraný programovací jazyk mohou pomoci při komunikaci bota s webovými stránkami na internetu, databází a dalšími službami pro vlastní dovednosti bota.

3.2.1 Discord API

Discord API je preferovaný způsob komunikace mezi botem a platformou Discord. Umožňuje snadné, rychlé a bezpečné vytvoření bota pro komunikaci s platformou přes dva základní prvky: *WebSocket API* a *REST API*. Na obr. 4 převzatého z článku [7] je vidět průběh komunikace, kdy platforma vlevo komunikuje ve směru bota pomocí *WebSocket API* [6], a bot vpravo ve směru platformy pomocí volání *REST API*. [7]



Obrázek 4: Způsob komunikace bota a platformy [7]

3.2.1.1 WebSocket API

WebSocket API je používán při přijímání eventů, při změně stavu na platformě, ze serverových strojů platformy Discord. Komunikace je navázána mezi platformou a botem skrze websocket, který je jednou vytvořen a umožňuje přenášet informace v „realtime“, eliminuje se tím potřeba pro každý přenos informace vytvořit nové spojení a vzniká tak velmi rychlý způsob přenosu informací.

Mezi eventy, které se přenášejí z platformy botu jsou například: uživatel zaslal zprávu, uživatel upravil zprávu, uživatel smazal zprávu, uživatel se připojil ke komunitě, uživatel připnul zprávu, uživatel vytvořil vlákno ve zprávě, uživatel se připojil k hovoru, uživatel si ztlumil mikrofon v hovoru.

Během přenosu informace o eventu jenž nastal, jsou spolu s ním předávány objekty *zprávu*, *uživatele*, *místnosti*, *komunitního serveru* obsahující jejich úplné informace. Po úspěšném přenosu informace o tom, že nastal event, si bot ukládá všechny informace o objektech do *cache*, se kterými je poté schopen pracovat své pro úkony a odesílá odpovědi s výstupy do *REST API*.

3.2.1.2 REST API

Akce, které je bot schopen provést, musí učinit přes volání *REST API*. Bot je také schopen si skrze *REST API* vyžádat získání informací, ale to není zpravidla kvůli efektivitě přenosu používáno, neboť většinu potřebných informací v aktuálním stavu, již dostal z websocketu a má je uložené ve své *cache*.

Bot z této cache vytáhne objekty, se kterými potřebuje pracovat a při provedení změn nad nimi odesílá do *REST API* pouze *identifikátor* objektu a změnu, kterou provedl. Neodesílá tak znovu celý objekt z důvodu rychlosti a případnému zahlcení API.

Identifikátor objektu je celé přirozené číslo, pod kterým si platforma je schopna najít objekt, kterému patří. V případě bota si i bot je schopen pomocí identifiká-

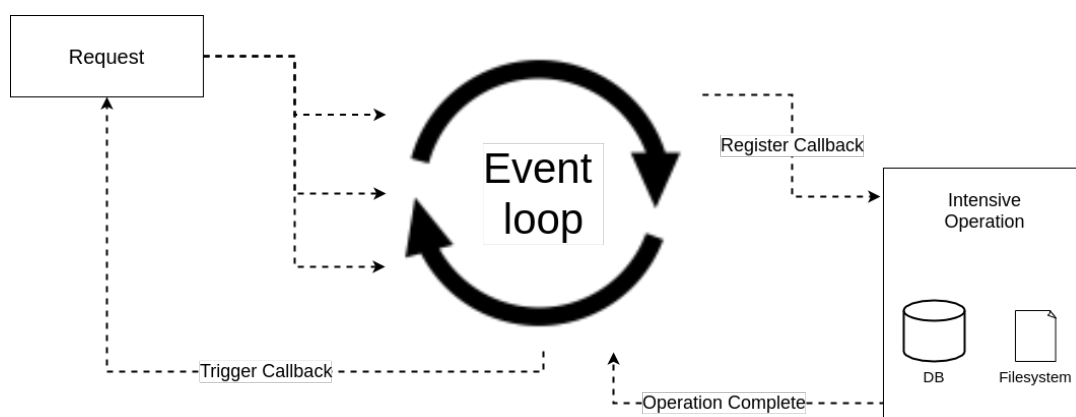
toru dohledat objekt, kterému patří, ale s omezením, že objekt se musí nacházet v jeho cache. Nemůže tak například získat objekt komunitního serveru jehož bot není členem i kdyby znal jeho identifikátor, protože takový objekt mu websocket s tímto identifikátorem nikdy nezaslal.

3.2.2 Asyncio

Asyncio je knihovna pro práci s asynchronním programováním používaném při komunikaci klient-server, kdy nějaký čas trvá než se informace přenesou a dostaví se její odpovědi, pro využití procesorového času efektivně namísto čekání se vykonává něco jiného zatímco se čeká než odpověď dorazí. [8] Asyncio, přesněji asyncIO je zkratka za asynchronous input/output, technologie která se stará o efektivní hospodaření s procesorovým časem při operacích, jenž mají delší efekt jako právě komunikace po síti.

Tato technologie dává možnost tzv. „single threaded“ procesům zdánlivě vykonávat více operací současně. Dlouho trvající operace nejprve všechny naráz jednu po druhé spustí a po vyčkání vrací v žádaném pořadí odpovědi po provedení operace, než aby se sekvenčně spustila jedna operace zvlášť a vyčkalo se na její dokončení než se spustí v sekvenci další dlouhotrvající operace.

Program má pouze jedno vlákno označované též jako *main thread* a tedy jeden *execution stack*, kde může vykonávat pouze jednu operaci. Při asynchronním programování požadavek jako je získání dat z webové stránky by blokoval *execution stack* po celou dobu čekání na odpověď. Namísto toho předává tento svůj požadavek do *event loopu* a z *execution stacku* se tím funkce odstraní a na její místo se ukládá návratová adresa, kam se tato funkce po vykonání *event loopem* navrátí. Funkce je tedy *callback* funkcí. *Event loop* se postará o obsluhu tohoto požadavku a spustí jej. Po dokončení požadavku se skrze *callback* vrací odpovědi požadavku zpátky do *execution stacku* *main threadu*.



Obrázek 5: Asynchronní programování s jedním procesem [13]

Na obr. 5 vlevo vidíme request z *main threadu* programu předaný *event loop* smyčce starající se o časově náročné input/output operace znázorněné vpravo,

po jejich dokončení se vrací výsledek operace zpět přes callback do main threadu programu.

3.2.3 FFmpeg

Je open-source software [CLI](#) nástroj pro práci s audiem a videem a dalšími multimediálními soubory a datovými proudy. [14] [15] *Fast Forward moving picture experts group*, zkráceně FFmpeg byl vytvořen již v roce 2000 a dnes je používán ve spoustě aplikací jako je *Google Chrome*, *blender* a platformem jako *YouTube*, *Discord* a *Vimeo*. Nástroj je schopen dekodovat, enkodovat, transkodovat, multiplexovat, demultiplexovat, streamovat, filtrovat a přehrávat většinu multimediálních souborů na světě. Má podporu přes 100 různých video kodeků.

FFmpeg funguje tak, že vezme jakýkoliv multimediální soubor, který pomocí demultiplexeru rozdělí do několika audio a video stop jako separátní data pakety. Pakety jsou dále dekodovány do jednotlivých snímků, které poté mohou být zpracované či filtrované. Zpracováním jako například úpravou jasu, kontrastu, přidáním titulků. Poté tyto snímky jsou zpátky zakódovány a multiplexerem složené nazpět do cíleného formátu.

Součástí nástroje je také nástroj *ffplay* pro přehrávání multimediálního souboru a *ffprobe* pro získání metadat ze souboru.

3.3 Knihovny pro práci s Discord API

K funkcionalitám platformy Discord sice jde přistupovat napřímo pomocí API v různých verzích a použitím [HTTP](#) protokolu a zasíláním *requestů*, ale knihovny mohou tuto činnost podstatně zjednodušit a usnadnit a použít ji je zkrátka výhodnější. Každá knihovna zapouzdřuje všechna volání do Discord API a stará se o limity a chybové stavy.

Limit, o který se stará, je *ratelimit* [4] a chrání uživatele knihovny před jeho překročením. Pokud kód provádí spoustu požadavků rychle tak předchází zahlcením API pozdržejím požadavků překračující určitý limit a odesílá je postupně v pořadí co nejdříve to bude opět možné.

Zamezuje odeslání informací, které by vedly k chybovým stavům API a tyto špatné informace propaguje knihovna svými chybovými stavy v příslušném programovacím jazyce.

Discord sám některé z knihoven třetích stran [12] schválil a označil, že splňují požadavky pro jejich API, výrazně také doporučil použití takových knihoven pro komunikaci s platformou než přístupem napřímo. Nejčastěji používané knihovny vidíme v tabulce 4 níže:

| Název knihovny | Programovací jazyk |
|----------------|--------------------|
| Discord.Net | C# |
| discord.js | JavaScript |
| discord.py | Python |
| Discordia | Lua |

Tabulka 4: Knihovny pro práci s boty [12]

3.3.1 Knihovna discord.py

Jedná se o knihovnu pro Discord API v programovacím jazyce Python, vytvořenou jako open-source projekt se zdrojovým kódem na platformě Github. [9] Team, který knihovnu vytvořil ji udržuje aktuální, funkční a snaží se pokrýt vše, co nabízí celé Discord API a vytvořit tak jednoduchý, rychlý a bezpečný nástroj při vývoji a implementaci botů. Uživatelé knihovny zcela abstrahuje od volání do zmíněného API, který v kódu převážně pracuje pouze objektově orientovaným přístupem. Tento způsob dělá kód jednodušším, čitelnějším a přenositelnějším.

Nově přidané funkce platformou Discord a Discord API bývají implementované do knihovny v řádu dnů, neboť se na jejím vývoji podílí přes 300 kontributorů.

Knihovna je ve dvou verzích, jedna bez podpory audia a druhá s podporou audia. Při její instalaci je tak nutné specifikovat, kterou nainstalovat. Bez explicitního uvedení o audio verzi je nainstalována základní obsahující eventy (zasílané websocketem) a podpora pro vytváření příkazů.

3.4 Možnosti hostování aplikace

Spuštění může být realizováno na jakémkoliv hardware běžícím s libovolným operačním systémem a nainstalovanými potřebnými programy na osobním počítači, osobním mikropočítači nebo ve službě cloudu na pronajatém vzdáleném počítači.

Použití osobního počítače je způsob nejčastěji používaný jen při vývoji a testování aplikace bota a to z několika důvodů. Je vhodné a rychlé vyzkoušet si všechny implementované změny ihned bez nutnosti přenosu souborů na jiný hardware. Avšak bývá nežádoucí ponechávat neustále zapnutý a připojený k síti osobní počítač, proto se na ostrý provoz zpravidla vybírá plnohodnotný hosting.

Použití osobního mikropočítače jako je Raspberry Pi [10] je plnohodnotná varianta k hostování aplikace bota. Raspberry Pi navíc dává velikou flexibilitu v operačních systémech, hardwarových specifikacích osobního mikropočítače a jeho provozování je levné.

Cloudová služba AWS Cloud Computing [11] dává možnost se u jejich služby zaregistrovat a pronajmout si vzdálený stroj na kterém bot následně poběží. Nabídka vzdálených strojů a jejich hardwarová konfigurace je veliká, pro nenáročné aplikace postačujícím základní výpočetní výkon jsou tyto stroje i bezplatné.

4 Návrh funkcionalit

Tato kapitola se zabývá návrhem funkcionalit automaticky prováděných botem, případně funkcionalitami, které manuálně vyvolá uživatel, ještě před jejich implementací v kódu je detailně probrán popis činnosti těchto funkcionalit.

Automaticky prováděné úkony jsou takové, které reagují na event z websocketu neboli změnu stavu platformy. K těmto úkonům patří při připojení nového uživatele kontrola jeho záznamů, zaslání pravidel komunitního serveru a přiřazení komunitou předem vybrané role s pravomocemi.

K manuálním úkonům slouží příkazy, které uživatel může vyvolat pomocí textového pole pro odeslání zprávy do chatu. K těmto příkazům se řadí správa uživatelů – vyhoštění a zablokování s přidáním záznamu o uživateli do databáze, přesouvání skupiny uživatelů mezi hlasovými kanály. Správa událostí – vytvoření události s názvem, popisem, datem konání a evidencí přihlášených osob, kterým se těsně před počátkem události zašle upozornění. Přehrávání audio stop – přehrání jednoho multimediálního obsahu, přehrání více multimediálního obsahu dané playlistem, úpravou hlasitosti, zobrazení fronty, přeskočení aktuálně přehrávaného audia.

4.1 Automatické vlastnosti

Jedná se o takové vlastnosti bota, které jsou prováděny automaticky bez zásahu uživatele, uživatel může pouze nadefinovat jejich chování pro daný komunitní server. K takovým vlastnostem se řadí informace o změně stavu na platformě jako je připojení uživatele *on_member_join*, připojení klienta ke komunitnímu serveru *on_guild_join*, při zaregistrování nové zprávy *on_message*, při zaregistrování upravené zprávy *on_message_edit*, při zaregistrování úspěšné inicializace klienta do platformy *on_ready*.

Pro typ *on_member_join* se provede získání pravidel komunitního serveru a zaslání nově připojenému uživateli, získání přidělené role s pravomocemi a následné přidělení role uživateli, zjištění existence záznamů o trestech uživatele a následné upozornění moderátorů komunitního serveru.

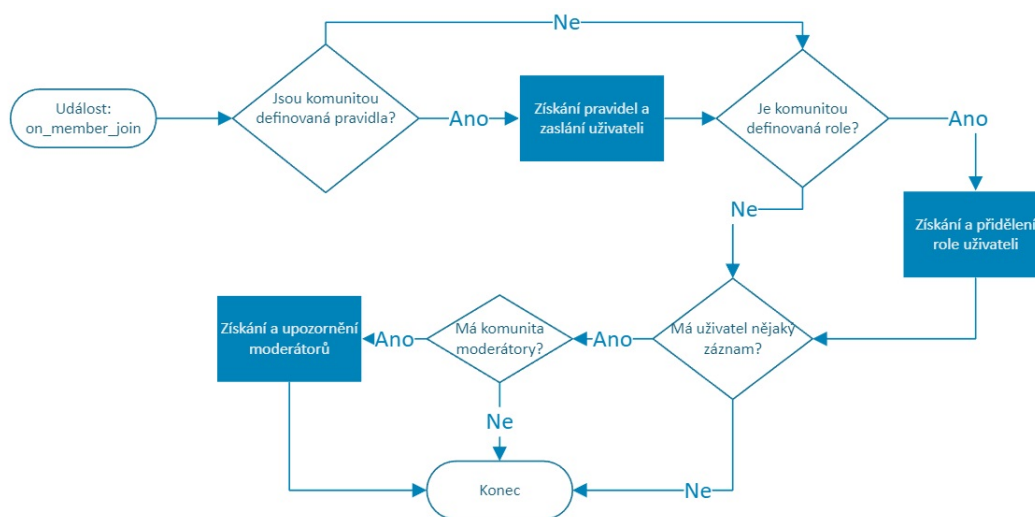
Pro typ *on_message* a *on_message_edit* provede získání slovníku zakázaných slov a frází, zjištění zda zpráva obsahuje některé z těchto slov a frází, pokud ano je zpráva okamžitě odstraněna.

Typy *on_guild_join* a *on_ready* slouží k uvedení bota do provozu pro daný komunitní server, přidává záznam o serveru do databáze. Tuto automatickou vlastnost nemůže uživatel komunity nijak definovat podle svých vlastních potřeb.

4.1.1 Událost: nový člen

Při získání eventu z websocketu týkajícího se připojení nového člena jsou obslouženy všechny tři vlastnosti zároveň: zaslat pravidla, přiřadit role a pravomoce, zkontrolovat záznamy.

Reakce o změně tohoto stavu je rozdělena na každou činnost zvlášť a každá může být vykonána individuálně, je-li v komunitním serveru zadefinována. Člen s příslušným moderátorským oprávněním si musí v komunitním serveru definovat své pravidla, které chce, aby bot odesílal. Definovat roli, aby ji bot přiřadil a jmenovat členy, ke kterým se bot bude chovat jako k moderátorům komunity a upozorní je na uživatele se záznamy.



Obrázek 6: Obslužení události při příchodu nového uživatele

4.1.1.1 Zaslání pravidel

Pro automatické zasílání pravidel se alespoň jedno pravidlo musí definovat na komunitním serveru pomocí přidávacího příkazu *rules add*. Příkazem *rules reset* se smažou existující pravidla komunity. Pro zobrazení všech platných pravidel poslouží příkaz *rules*. Pravidla se automaticky číslovají v pořadí ve kterém byly přidány a jsou určeny k zaslání novému uživateli po jeho připojení do soukromé zprávy.

4.1.1.2 Přiřazení role, pravomocí

K přiřazování role novým členům je zapotřebí, aby již na komunitním serveru přes uživatelské rozhraní byla role nadefinována. Poté přes příkaz jako *autorole set* může moderátor přiřazovanou roli s pravomocemi nastavit. Příkazy *autorole show*, *autorole remove* ji zobrazí nebo smažou. Takto definovaná role je automaticky přiřazena novému uživateli po jeho připojení na komunitní server za podmínky, že je v hierarchii pod rolí bota.

4.1.1.3 Kontrola záznamů

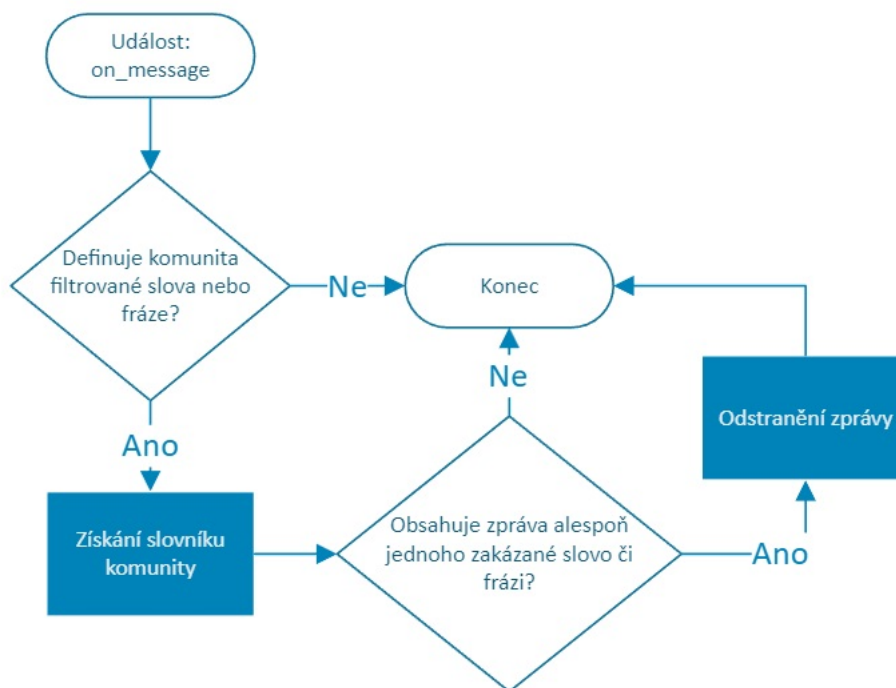
Kontrola záznamu se provede automaticky poté, co se uživatel, který má záznam v rejstříku, připojí na server. Upozornění dostanou pouze předem jmenovaní

moderátoři, které lze přidat, odstranit a zobrazit příkazy *moderation add*, *moderation reset*, *moderation show*. Možnost zkontrolovat nový záznam již stávajícího člena pomocí příkazu *records show*. Záznamy do rejstříku udělají příkazy pro vyhoštění či zablokování uživatele v libovolné komunitě (i mimo stávající), kterými jsou *kick* a *ban*.

4.1.2 Událost: zpráva

Zaslaný event o změně stavu, že je zaslána či upravena zpráva, bot musí zkontrolovat obsah zprávy a v případě závadné zprávy takovou zprávu nechat odstranit.

Následná reakce na změnu tohoto stavu obsahuje jednu činnost, zkontrolovat slovník slov a frází v příslušném komunitním serveru nově příchozí (upravené) zprávy při jejím zpracování.



Obrázek 7: Obsloužení události nové (upravené) zprávy

Na obr. 7 je vidět událost při zjištění stavu o zaslání nové zprávy, kdy se získá „blacklist“ slovník komunity a pokud ve zprávě existuje alespoň jeden výskyt ze slovníku, je taková zpráva odstraněna. Pro událost při upravené zprávě by vypadal diagram stejně jako na obr. 7, s jediným rozdílem a to spouštěcí událostí *on_message_edit*.

4.1.2.1 Kontrola odeslaných a upravených zpráv

Pro automatickou moderaci chatu botem je potřeba přes příkaz *filter add* definovat alespoň jedno zakázané slovo (či frázi) v komunitním serveru. Příkazem

filter remove je možné odstranit konkrétní napsané slovo či fráze, *filter show* zase zobrazí veškerý slovník. Není žádoucí, aby uživatel bez oprávnění mohl příkaz pro zobrazení spustit a nalézt případné chyby k obcházení. Též se nerozlišuje, zda slovo či fráze má malá nebo velká písmena.

4.2 Manuální vlastnosti

U manuálních vlastností bota se jedná o takové akce, které pro jejich vykonání musejí být vykonány uživatelem, tedy uživatel musí spustit příkaz přes kontextové menu obr. 3 lomítkového příkazu nebo napsáním prefixované zprávy obsahující validní název příkazu obr. 2.

Vlastnosti, které tyto manuální příkazy mají zastat jsou pro správu uživatelů (vyhoštění, zablokování), správu událostí (vytvoření, smazání, editace), přehrávání audio stop (přehrání, přeskočení, zobrazení fronty, úprava hlasitosti).

Tyto příkazy musejí být manuální, neboť potřebují navíc vstup od uživatele k doplnění či upřesnění některých informací potřebných k jejich provedení.

4.2.1 Správa uživatelů

Pro správu uživatelů je zavedena možnost vyhostit jednoho či více uživatelů zároveň pomocí příkazu *kick* s nepovinným odůvodněním, jejich vyhoštění vede k vytvoření záznamu v databázi bota pro případné varování uživatelů v jiných komunitách o jejich incidentu. Podobně na tom je příkaz zablokování *ban* s rozdílem, že umožňuje navíc volbu smazání jejich výskytu v historii chatu za posledních několik předchozích dní.

Zprostředkuje pro majitele komunity s nejvyšším oprávněním možnost zasílat hromadně všem uživatelům jeho komunitního serveru zprávu do soukromé zprávy příkazem *admin message* (či *embedded*), dbá se na potencionální zneužití při napadení účtu majitele a je nastaven limit pro tuto funkcionalitu, aby nemohla být v případě velkých komunit zneužita např. pro scam či phishing.

Pro správu nad hlasovými kanály má moderátor s oprávněním možnost příkazem *move* vybrat „mentions“ připojených uživatelů a označením cílového hlasového kanálu, do kterého je chce následně přesunout.

4.2.2 Správa událostí

Zprostředkování a správa naplánovaných událostí je uděláno pomocí zapouzdřené zprávy a interakcí příslušnými tlačítky (*Accept*, *Decline*, *Tentative*, *Cancel*).

Tlačítko *accept* jde použít dokud se nepřekročí nastavený limit přihlášených uživatelů, je-li stanoven. *Decline* odmítne událost, *tentative* pro nerozhodného uživatele a tlačítko *cancel* použitelné pouze pro uživatele jenž událost vytvořil pro její následné zrušení.

Zapouzdřená zpráva umožňuje formátovat jednotlivé položky zprávy do řad a sloupců s podporou odkazů a obrázků. Vytvoření události se provede příkazem

event echat, který stojí za zkratkou event external chat (kvůli odlišení event internal), při jeho použití je uživatel vyzván k doplnění informací názvu, popisku, datu, volitelnému maximálnímu počtu přihlášených uživatelů a výběru zda bude použito číslo počtu přihlášených nebo jejich celé jméno (vhodné pro menší událost) a místa konání buď libovolného textu či hlasového kanálu. Pro odstranění *events echat* není nutné mít vlastní příkaz, neboť stačí zprávu moderátorem odstranit.

Vedle toho existuje příkaz *events ilocation* (popřípadě *ivoice*) pro tvorbu událostí integrovaných přímo v Discordu v komunitě. Tyto integrované události jdou upravovat pomocí *events edit_location* (*events edit_voice*) a smazat příkazem *events idelete*, za pomoci vyhledání první shody hledaného názvu události se nalezená událost odstraní.

4.2.3 Přehrávání audio stop

Přehrávání audio stop je zprostředkované příkazem *play*, který bota připojí do stejného hlasového kanálu jako uživatel jenž příkaz vyvolal a začne přehrávat hudbu danou odkazem, v odkazu může být pouze obsah z platformy YouTube.

Při opakovaném použití příkazu *play* se další audio zařazuje do fronty, která se dá zobrazit příkazem *queue*. Pro přehrávání playlistu slouží příkaz *play_playlist*, který zařadí každý přístupný obsah z playlistu do fronty. Příkaz *volume* umožní upravit hlasitost přehrávaného audia a upraví hlasitost i pro další přehrávaný obsah. Příkaz *pause* pozastaví přehrávání audia, příkaz *resume* opět spustí přehrávání pozastaveného audia, *stop* zastaví audio a odpojí bota od hlasového kanálu. Příkazy jako *next* přeskočí aktuálně přehrávajícím audio a začne přehrávat další v pořadí, *shazam* určí název aktuálně přehrávané skladby.

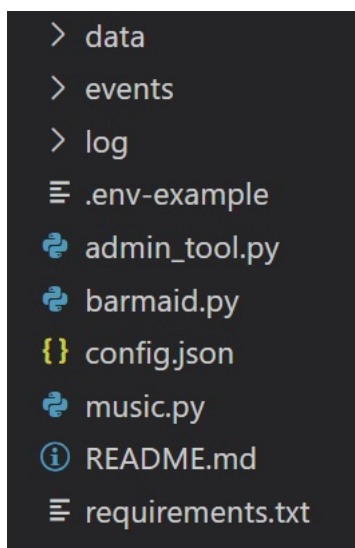
5 Implementace

Tato kapitola se zabývá, jakým způsobem je bot vyvinut a implementován.

Pro implementaci bota byla zvolena knihovna *discord.py*, která zajistí komunikaci bota s Discord API, knihovna *yt_dlp*, která umožní komunikovat s platformou YouTube a nepřímo použitá knihovna pro manipulaci s audiem *FFmpeg*. Knihovna *asyncio* umožňuje kód spouštět a vykonávat asynchronně. Jako další knihovny jsou použity *aiohttp* pro potřeby komunikace s REST API, *aiofiles* pro asynchronní přístup k souborům a knihovna *python-dotenv* sloužící jen pro uložení autorizačního tokenu mimo zdrojový kód za účelem bezpečnosti.

Je také implementována vlastní databáze za pomoci **JSON** formátu, neboť není nutné ukládat velké množství dat a vedle aplikace bota mít databázový systém by bylo paměťově náročnější než samotná aplikace bota.

5.1 Souborová struktura



Obrázek 8: Souborový systém aplikace bota

Na obr. 8 je souborová struktura rozdělena následujícím způsobem:

data

Složka obsahuje veškerou práci s daty se kterými bot nakládá, jako je načtení informací z konfiguračního souboru, hlavní databáze bota a databáze záznamů uživatelů, modul pro komunikaci nad zmíněnými databázemi.

events

Složka obsahuje moduly související s plánovanými událostmi.

log

Složka obsahuje modul pro vytvoření a obsluhu logů jenž bot využívá a soubor, do kterého se log zapisuje.

.env-example

V souboru `.env-example` se nachází placeholder pro autorizační token bota, který je potřeba doplnit při zprovoznění vlastního účtu bota. Soubor je poté nutné přejmenovat jako `.env` a ponechat.

admin_tool.py

Je modul obsahující všechny příkazy týkající se správy komunity, uživatelů.

barmaid.py

Je hlavní spouštěcí soubor bota, který připojí ostatní moduly a bota přihlásí do Discordu. Definuje také chování některých eventů websocketu, kterým bot naslouchá a obsluhuje je.

config.json

Je základní konfigurační soubor ve kterém se v případě vlastního bota musí vhodně nastavit údaje s identifikátorem bota, dále zde lze upravovat chování bota k příkazům.

music.py

Je modul obsahující všechny příkazy týkající se přehrávání audia.

README.md

Je soubor obsahující popis projektu a jeho instalaci, určený pro repozitář na Github platformu.

requirements.txt

Je soubor obsahující popis všech potřebných knihoven v příslušných verzích pro instalaci skrze *Python Installer Package*, zkráceně *pip*.

5.2 Spouštěcí soubor

Jako hlavní spouštěcí soubor je modul s názvem `barmaid.py`, který při spuštění interpretem připojí ostatní moduly, nastaví eventy na které bot bude reagovat z websocketu, připojí se k Discordu, bota autorizuje a spustí, spouštěcí modul rovněž definuje funkci pro správný chod prefixových příkazů.

```
1 # Intents manages some level of permissions bot can do
2 INTENTS = Intents.default()
3 INTENTS.members, INTENTS.presences = True, True
4 INTENTS.message_content, INTENTS.reactions = True, True
```

Zdrojový kód 1: Nastavení oprávnění bota v Discord API

Ve zdrojovém kódu 1 definujeme botu přístup ke členům komunitního serveru a jejich aktivit, obsahu zpráv a jejich reakcí.

```

1  async def get_prefix(client:commands.Bot, message:Message):
2      if not message.guild:
3          return commands.when_mentioned_or(S.DEFAULT_SERVER_PREFIX) (
4              client, message)
5
6      prefix = await read_db(DATABASE, message.guild.id, 'prefix')
7      if not prefix:
8          is_okay = await insert_db(DATABASE, message.guild.id, 'prefix'
9              , S.DEFAULT_SERVER_PREFIX)
10         if not is_okay:
11             return
12         prefix = S.DEFAULT_SERVER_PREFIX
13     return commands.when_mentioned_or(prefix) (client, message)

```

Zdrojový kód 2: Funkce určující prefix

Zdrojový kód 2 je funkce, která zajistí zjištění příslušného prefixu používaného v konkrétním komunitním serveru je-li nastaven, jinak se použije symbol „.“ jako výchozí.

```

1  # Client is top level var because of @ decorator for events
2  CLIENT = commands.Bot(command_prefix=get_prefix, intents=INTENTS)
3
4  # Create of log handle
5  log_handle=logging.Logger = setup_logging()

```

Zdrojový kód 3: Vytvoření bota

Kód 3 zajišťuje vytvoření instance třídy *commands.Bot* za použití prefixové funkce a oprávnění, které bot bude mít definované z předchozích zdrojových kódů 1 a 2. Také vytváří proměnnou *log_handle*, pomocí které lze zapisovat log při běhu aplikace.

```

1  @CLIENT.event
2  async def on_command_error(ctx:Context, error:commands.
3      CommandError):
4      if isinstance(error, commands.CommandNotFound):
5          await ctx.send(error, ephemeral=True)
6      return

```

Zdrojový kód 4: Ukázka callback funkce websocket eventu

Kód 4 definuje callback funkci, která bude reagovat na události z websocketu, pro ukázkou je zde uvedena pouze jedna, ale spouštěcí soubor obsahuje těchto funkcí více: *on_ready*, *on_guild_join*, *on_member_join*, *on_message*, *on_message_edit*, *on_message_error*. Tyto funkce budou ještě popsány v implementaci dále.

Zdrojový kód 7 spustí bota pomocí autorizačního tokenu, který nalezne v souboru `.env` a následně spustí funkci 6 pro načtení ostatních modulů do instance třídy bota v proměnné `CLIENT`. Kód 5 je spuštěn poté co je bot úspěšně nastartován a dorazí jeden z prvních eventů websocketu k synchronizaci lomítkových příkazů, slouží k tomu funkce `setup_hook`. Zároveň s tím nastavuje třídu `EventView`, o té ještě později, jako persistentní vůči restartu, aby tlačítka jenž bot vytváří fungovali i po jeho vypnutí a následném zapnutí.

```
1 @CLIENT.event
2 async def setup_hook():
3     synced_commands = await CLIENT.tree.sync()
4     print(f"In-app commands have been synchronized.")
5     CLIENT.add_view(EventView())
```

Zdrojový kód 5: Synchronizace příkazů

```
1 # Listed modules to be loaded
2 EXTENSIONS = [
3     "admin_tool",
4     "events.event",
5     "music",
6 ]
7
8 async def install_extensions(target: commands.Bot):
9     for ext in EXTENSIONS:
10         await target.load_extension(ext)
```

Zdrojový kód 6: Instalace modulů

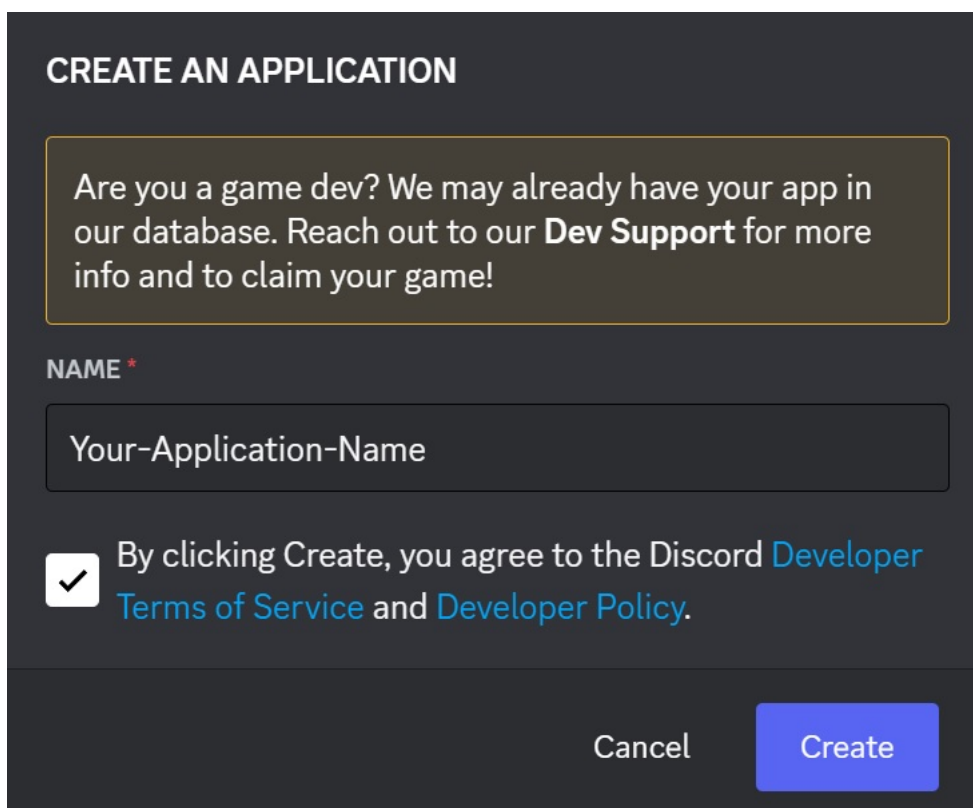
```
1 if __name__ == "__main__":
2     load_dotenv()
3     CONNECTION_TOKEN = os.getenv("TOKEN")
4
5     loop = asyncio.new_event_loop()
6
7     loop.run_until_complete(install_extensions(CLIENT))
8     CLIENT.run(CONNECTION_TOKEN)
```

Zdrojový kód 7: Spuštění bota

5.3 Vytvoření aplikace a účtu jako bot

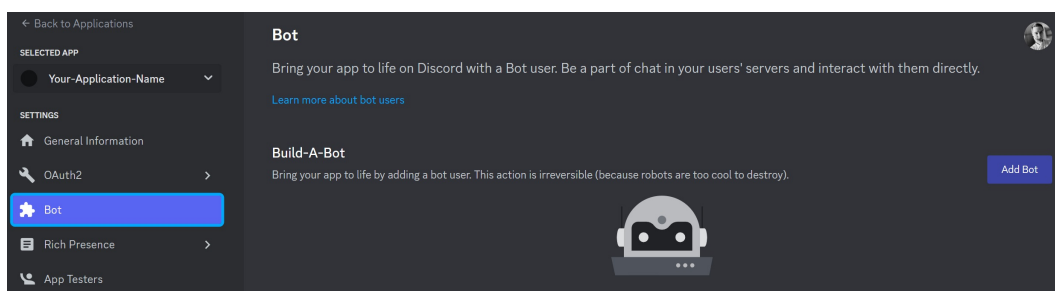
Vytvoření aplikace probíhá po přihlášení hlavním účtem uživatele na webové stránce platformy a navštívením webové adresy `discord.com/developers/applications`,

zde je v pravém horním rohu tlačítko „Create Application“ a vyzvání k výběru jména aplikace viz obr. 9, toto jméno bude jménem účtu bota.



Obrázek 9: Vytvoření aplikace

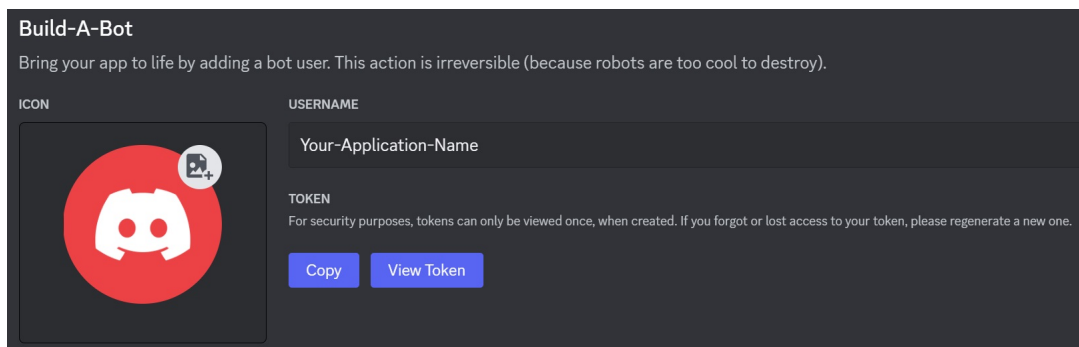
Po vytvoření aplikace je nutné přejít do sekce „Bot“ a vytvořit nového bota tlačítkem „Add Bot“ viz obr. 10.



Obrázek 10: Vytvoření aplikace jako bot

Po vytvoření bota se zpřístupní účet bota a je možné zobrazit jeho autorizační token jednorázovým tlačítkem „View Token“ viz obr. 11. Tento token slouží ke spuštění bota hlavním spouštěcím souborem, token je nutné vložit do souboru *.env-example* na příslušné místo a přejmenovat soubor na *.env*.

V sekci „General Information“ na webové stránce lze, viz obr. 12, nalézt *application id*, kterým je nutné příslušně vyplnit soubor *config.json*.



Obrázek 11: Získání autorizačního tokenu bota

| |
|---|
| NAME |
| Your-Application-Name |
| DESCRIPTION (MAXIMUM 400 CHARACTERS) Your description will appear in the About Me section of your bot's profile. |
| TAGS (MAXIMUM 5) Add up to 5 tags to describe the content and functionality of your application. |
| APPLICATION ID 1088846590430674944 |
| Copy |

Obrázek 12: Identifikátor aplikace

5.4 Ukládání dat

Ukládání informací je nutné, aby bot mohl rozlišovat data týkající se pouze konkrétního komunitního serveru. Ukládá tedy pro každý komunitní server jeho prefix, pravidla, roli pro přidělení a veškerou organizaci naplánovaných událostí. Dále používá globální databázi pro záznamy uživatelů, jenž byly z některého serveru vyhoštěni či z nějakého důvodu zablokováni.

Pro dva nezávislé soubory s daty *user_records.json* a *data.json* je zaveden univerzální přístup modulem *jsonified_database.py*, který umožňuje čtení a zápis do těchto souborů na základě výběru.

K otevření cíleného souboru slouží kód 8, který zajišťuje asynchronní přístup k souboru a vrátí jeho celý obsah ve formátu **JSON**.


```

1  async def open_file(file_name:str) -> dict:
2      async with aiofiles.open(file_name, READ_FLAG) as f:
3          contents = await f.read()
4          json_as_dict = json.loads(contents)
5      return json_as_dict

```

Zdrojový kód 8: Otevření databázového souboru

Pro zápis do cíleného souboru slouží funkce *flush_file* ze zdrojového kódu 9, který zajišťuje asynchronní přístup k souboru a zapisování do něj.

```

1  async def flush_file(file_name:str, data:dict) -> str:
2      async with aiofiles.open(file_name, WRITE_FLAG) as f:
3          await f.write(data)
4      return data

```

Zdrojový kód 9: Zápis do databázového souboru

Za pomoci těchto dvou základních univerzálních funkcí jako je čtení a zápis jde dále vytvořit funkci, která přečte danou hodnotu podle klíče v databázi, slouží k tomu funkce *read_db* viz zdrojový kód 10.

```

1  async def read_db(file_name:str, id:int, key:str):
2      id_str = str(id)
3      data = await open_file(file_name)
4
5      try:
6          value_by_key = data[id_str][key]
7          return value_by_key
8      except KeyError as e:
9          return None

```

Zdrojový kód 10: Čtení v databázovém souboru

Základní funkcí aktualizace dat v databázových souborech slouží funkce *update_db* viz zdrojový kód 11, dále modul *jsonified_database.py* obsahuje i další funkce jako *insert_db*, *delete_from_db*, *add_id* a *id_lookup*.

```

1  async def update_db(file_name:str, id:int, key:str, new_value):
2      id_str = str(id)
3      data = await open_file(file_name)
4
5      try:
6          data[id_str][key] = new_value
7      except KeyError:
8          return None
9
10     new_data = json.dumps(data, indent=2)
11     result = await flush_file(file_name, new_data)
12     return result

```

Zdrojový kód 11: Aktualizace dat v databázovém souboru

5.5 Struktura příkazů v kódu

Struktura příkazu je demonstrována a detailněji popsána na příkladu jednoho z příkazů. V kódu 12 je vidět struktura v kódu. Řádek č. 1 určuje pomocí dekorátoru o jaký příkaz se jedná, `@commands.command()` definuje prefixovaný příkaz, `@commands.hybrid_command(with_app_invoke=True)` definuje prefixový příkaz a k němu stejnojmenný lomítkový příkaz na základě pravdivostní hodnoty argumentu, `@commands.hybrid_group()` pak podobně jako `hybrid_command` definuje hlavní příkaz a k němu podpříkazy, lze později vidět v implementaci plánovaných událostí v podkapitole. Ve stejném dekorátoru lze definovat jméno příkazu, není-li definováno pak je použito jako jméno název definované funkce na řádce č. 6, a případné „aliasy“ pouze pro prefixované příkazy.

Řádek č. 4 určuje rozsah platnosti příkazu, dekorátor `@commands.guild_only()` určí příkaz pouze pro použití uvnitř komunitního serveru, není-li definováno pak příkaz je platný i ve soukromé zprávě s botem.

Dekorátor `@commands.has_guild_permissions()` na řádce č. 5 ověřuje, zda uživatel jenž příkaz vyvolal má dostatečná oprávnění, pokud nemá je programem vyvolána výjimka, která může být buď obsloužena pomocnou funkcí se stejnojmenným dekorátorem jako je název příkazu a vlastností `error`. Pro tuto funkci `clear` by to bylo například `@clear.error` a funkce s libovolným jménem např. `clear_error`, pro příklad jak taková chybová funkce může vypadat viz. zdrojový kód 13. Není-li taková funkce definována, je pak odchycena eventem websocketu s názvem `on_command_error`.

Řádek č. 6 definuje funkci, která se vykoná při zavolání příkazu za pomoci prefixu/lomítka dle příslušného dekorátoru u funkce, funkce musí mít vždy jeden argument `ctx` třídy `commands.Context`, který obsahuje všechny informace z cache bota a lze pomocí tohoto objektu zjistit autora příkazu, komunitní server a na základě toho rozlišovat chování příkazu. Zbytek argumentů je na vývoji daného příkazu. Při použití statického typování pak u lomítkových příkazů dochází

```

1  @commands.hybrid_command(name="clear",
2     aliases=["clr", "delmsgs", "delmsg"],
3     with_app_command=True)
4  @commands.guild_only()
5  @commands.has_guild_permissions(manage_messages=True)
6  async def clear(ctx: commands.Context, amount: int=1):
7     """Removes any number of messages in chat history.
8
9     Args:
10        ctx: Current context of the message that invoked the command.
11        amount (optional): Number of messages to delete.
12    """
13    # If invoked via slash commands, no need to purge call itself.
14    await ctx.defer(ephemeral=True)
15    if not ctx.interaction:
16        amount += 1
17
18    if amount > 0:
19        await ctx.channel.purge(limit=amount)
20        amount = amount - 1 if not ctx.interaction else amount
21        await ctx.send(content=f"Cleared `{amount}` message(s)")
22    else:
23        await ctx.send(f"Amount of messages to be deleted has" \
24            " to be a positive number")

```

Zdrojový kód 12: Ukázka struktury příkazu v kódu, příkaz *clear*

ke správné nápovědě, co může do argumentu vstupovat. Je-li uvedena výchozí hodnota argumentu pak se tento argument stává nepovinným pro příkaz.

Zabalené řádky č. 7-12 obsahují dokumentaci funkce, jenž je pak následně použita pro popisky příkazů a jejich argumentů, je-li příkaz volán jako lomítkový.

První řádek v těle funkce č. 14 slouží k obslužení příkazu hybridně, buď interakcí skrze lomítkový příkaz nebo prefixovanou zprávu. Pokud se jedná o lomítkový příkaz, je nutné odpovědět pomocí *ctx.send()* do deseti vteřin. Trval by výpočet délky, pak vyprší timeout interakce a *ctx.send()* vyvolá výjimku, tento řádek zvýší timeout interakce z deseti vteřin na 15 minut. Pokud příkaz byl vyvolán pomocí prefixové zprávy, pak tento kód řádku č. 14 nemá žádný efekt a neprovádí nic s navýšením timeoutu. Pouze v obou případech může definovat skrze *ephemeral=True* viditelnost odpovědi pouze autorovi, co příkaz vyvolal.

Řádky č. 15 až 24 obsahují zbytek těla funkce a chování celého příkazu, v tomto příkladu smazání určitého počtu zpráv v textovém kanále, jenž byl příkaz spuštěn.

```

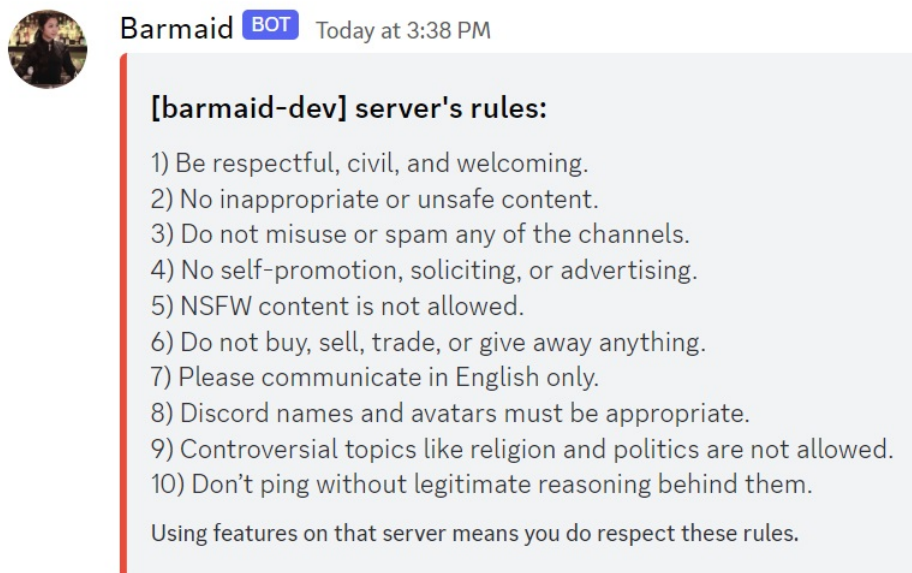
1  @clear.error
2  async def clear_error(ctx:commands.Context, error:errors):
3      # Bot missing permissions
4      await ctx.defer(ephemeral=True)
5      if isinstance(error, discord.Forbidden):
6          await ctx.send(f"{CLIENT} missing permissions `Manage
              Messages`",
7                          delete_after=S.DELETE_COMMAND_ERROR)
8      # User missing permission
9      elif isinstance(error, commands.MissingPermissions):
10         await ctx.send(error, delete_after=S.DELETE_COMMAND_ERROR)
11
12     if not ctx.interaction:
13         await delete_command_user_invoke(ctx, S.DELETE_COMMAND_ERROR
        )

```

Zdrojový kód 13: Chybová funkce k příkazu *clear*

5.6 Pravidla komunit

Pravidla komunit spadají pod správu a nacházejí se v modulu *admin_tool.py*. Zde existují příkazy *rules* pro zobrazení pravidel, *addrule* pro přidání nového pravidla ke stávajícím, *delrule* pro smazání konkrétního pravidla, *rules_reset* pro odstranění všech platných pravidel.



Obrázek 13: Ukázka zasláných pravidel novým členům

Pro zprovoznění automatického zasílání pravidel botem musí existovat alespoň jedno pravidlo zadané uživatelem s potřebnými moderátorskými pravomocemi. Pokud role uživatele nesplňuje příslušná oprávnění, jsou mu některé

z těchto příkazů nepovoleny a pouze napíší, které pravomoce jsou potřeba k jejich spuštění. K automatickému zaslání slouží callback funkce `on_member_join` reagující na event websocketu v hlavním spouštěcím souboru `barmaid.py`, tuto callback funkci také lze vidět ještě v kapitole týkající se kontroly uživatelů. Zde je uvedena pouze její část, týkající se pravidel, tedy přenechání na funkci `send_guild_rules`.

```
1 @CLIENT.event
2 async def on_member_join(member:Member):
3     # ... get role id from database and find corresponding role
4     await member.add_roles(role_to_give)
5
6     await send_guild_rules(member, member.guild)
7
8     if await check_records(member):
9         await aware_of_records(member, member.guild)
```

Zdrojový kód 14: Event (websocket) připojený člen

```
1 async def send_guild_rules(member:Member, guild_joined:Guild):
2     # ... get rules from database, format them
3     emb = Embed(title=f"[{guild_joined.name}] server's rules:",
4                 description=formatted_output,
5                 color=Colour.red())
6     emb.set_footer(text="Being part of that server means you do
7                 respect these rules.")
8     await member.send(embed=emb)
```

Zdrojový kód 15: Zaslání pravidel komunity

Event websocketu `on_member_join` popsany callback funkcí 14 je spuštěn při připojení nového člena do serveru a funkce přiděluje pravomoce, o tom v pozdější kapitole. Přenechává na funkci `send_guild_rules` zaslání připojenému členovi příslušná pravidla serveru. Nejsou-li stanovena žádná pravidla předem, není zasláno nic.

Funkce 15 obstarává zaslání uživateli, který se připojil, příslušná pravidla serveru, která se pokusí nalézt v databázi, naformátuje jejich výstup a zapouzdří je do zprávy, kterou mu následně zašle do soukromého chatu.

Zdrojový kód 16 slouží k zobrazení pravidel příslušné komunity, pravidla získává z databáze bota a formátuje každá pravidlo na vlastní řádek s řadovou číslovkou. Pokud žádná pravidla ještě neexistují, informuje o tom chybovou hláškou, taktéž odstraní prefixovou zprávu pokud ta příkaz vyvolala.

```

1  @commands.hybrid_command(with_app_command=True)
2  @commands.guild_only()
3  async def rules(ctx:commands.Context):
4      # ... handle slash command or prefix invoke, permissions
5
6      # ... find rules, format them
7      if guild_rules:
8          await ctx.send(formated_output,
9                          delete_after=S.DELETE_MINUTE)
10         return
11         await ctx.send("No rules have been set yet.",
12                         delete_after=S.DELETE_COMMAND_INVOKE)

```

Zdrojový kód 16: Zobrazení pravidel

```

1  @commands.hybrid_command(with_app_command=True)
2  @commands.guild_only()
3  async def addrule(ctx:commands.Context, *, new_rule:str):
4      # ... handle slash command or prefix invoke, permissions
5
6      # ... check database for already existing rules, if so
7      # then appends new to them
8
9      updated = await update_db(DATABASE,
10                               ctx.guild.id, "guild-rules", rules)
11      if updated:
12          await ctx.send("New rule applied.")
13          return
14      await database_fail(ctx)

```

Zdrojový kód 17: Přidání pravidla

K přidání nového pravidla slouží zdrojový kód [17](#), který nejdříve zkontroluje dostatečná oprávnění uživatele, získá stávající pravidla a k nim připojí pravidlo nové. Poté aktualizuje záznam v databázi a oznámí úspěšné zaznamenání pravidla. Pokud něco selže, je o tom uživatel informován chybovou hláškou. Pokud byla k vyvolání příkazu použita prefixová zpráva, bude odstraněna.

O vyresetování všech pravidel se postará zdrojový kód [18](#), který zkontroluje potřebné pravomoce, odstraní prefixovanou zprávu jenž příkaz vyvolala (pokud byla použita), odstraní záznam z databáze a informuje o úspěšném či neúspěšném vykonání.

```

1  @commands.hybrid_command(with_app_command=True,
2                             name="reset-rules")
3  @commands.guild_only()
4  async def rules_reset(ctx:commands.Context):
5      # ... handle slash command or prefix invoke, permissions
6
7      reset = await delete_from_db(DATABASE,
8                                   ctx.guild.id, "guild-rules")
9      if reset:
10         await ctx.send("Rules no longer apply.",
11                        delete_after=S.DELETE_COMMAND_INVOKE)
12         return
13     await database_fail(ctx)

```

Zdrojový kód 18: Resetování pravidel

Zdrojový kód 19 smaže pravidlo na příslušném indexu, příkaz zkontroluje potřebné pravomoce uživatele jenž příkaz vyvolal. Pokud byla použita prefixová zpráva k jejímu vyvolání dojde k jejímu smazání, odstraní se konkrétní pravidlo na daném indexu v databázi a dojde k informování uživatele o provedení, ať úspěšném nebo neúspěšném.

```

1  @commands.hybrid_command(with_app_command=True)
2  @commands.guild_only()
3  async def delrule(ctx:commands.Context, index:int):
4      # ... handle slash command or prefix invoke, permissions
5
6      # ... find rules in database
7      if not guild_rules:
8         await ctx.send("There are no rules therefore nothing to
9                        delete.")
10         return
11     del guild_rules[str(index-1)]
12     idx = 0
13     new_dict = {}
14     for _, value in guild_rules.items():
15         new_dict[str(idx)] = value
16         idx += 1
17     # ... update database

```

Zdrojový kód 19: Smazání pravidla

5.7 Kontrola zpráv

O kontrolu zpráv se stará hlavní spouštěcí modul, neboť v něm jsou definované všechny websocket eventy, na které bot reaguje a naslouchání zprávám je také

event. Jako eventy pro kontrolu zpráv slouží `on_message` a `on_message_edit`, kde v těle těchto callback funkcí je zavolána funkce `check_blacklist`, která se stará o zkontrolování zablokovaných slov v příslušné komunitě a dojde-li ke shodě je příslušná zpráva odstraněna. Z modulu ke správě `admin_tool.py` pomáhají k definování zakázaných slov příkazy `filter show`, `filter add`, `filter remove` pro zobrazení všech zablokovaných slov nebo frází, přidání/odebrání nového slova či fráze.

Ve zdrojovém kódu 20 dojde i ke kontrole vlastní zprávy zasláné botem, tuto skutečnost bot ignoruje a má pro sebe výjimku. Dochází ke kontrole, zda zpráva byla poslána v komunitním serveru. Pokud ano předává se zpráva ke kontrole příslušnému serveru funkcí `check_blacklist` viz kód 21, dále posílá zprávu ke zpracování prefix příkazů, které se mohou potencionálně ve zprávě nacházet. V neposlední řadě informuje uživatele, že bot nebude reagovat na interakce skrze soukromý chat s botem.

```
1 @CLIENT.event
2 async def on_message(msg:Message):
3     if msg.author == CLIENT.user:
4         return
5     if msg.guild:
6         await check_blacklist(msg.guild, msg)
7         await CLIENT.process_commands(msg)
8         return
9     await msg.channel.send("I don't serve any drinks here.")
```

Zdrojový kód 20: Event (websocket) nové zprávy

Kód 21 provádí smazání zprávy pokud zpráva obsahuje alespoň jedno slovo či frázi definovanou příslušnou komunitou jako nevhodnou, bot dává výjimku skrze `is_blacklist_exception` funkci pouze takové zprávě, která je zamýšlena k použití při odstraňování zablokovaného slova či fráze neboť tato zpráva bude takové slovo/frázi obsahovat.

Aby nebylo možné napsat nežádoucí slova „na podruhé“ obcházením funkcionality úpravy zpráv, tak je potřeba kontrolovat nežádoucí obsah i v nich, od toho slouží zdrojový kód 22, který je zjednodušenou verzí kódu při zaslání první zprávy `on_message` 20.


```

1  async def check_blacklist(guild:Guild, msg:Message):
2      if await is_blacklist_exception(msg):
3          return
4
5      # ... finds blacklist in the database
6
7      for w in blacklist:
8          if w in msg.content:
9              await msg.delete()
10             await msg.author.send(f"Word \"{w}\" is restricted to use
                in \"{guild.name}\"")

```

Zdrojový kód 21: Kontrola obsahu zprávy

```

1  @CLIENT.event
2  async def on_message_edit(before:Message, after:Message):
3      if after.author == CLIENT.user:
4          return
5      if after.guild:
6          await check_blacklist(after.guild, after)

```

Zdrojový kód 22: Event (websocket) upravené zprávy

5.8 Kontrola uživatelů, přidělení pravomocí

Kontrola uživatelů je zajištěna pomocí eventu websocketu *on_member_join* v hlavním spouštěcím souboru *barmaid.py*. Pro nastavení této funkcionality slouží příkazy z modulu *admin_tool.py* k nastavení moderátorů komunity a příkaz pro zobrazení záznamu konkrétního uživatele. Souvisejícími příkazy s touto funkcionalitou jsou *moderation show*, *moderation add*, *moderation reset*, *records*.

V předchozí kapitole již byl zmíněn zdrojový kód 14, kde je vysvětlena jedna část funkcionality týkající se pravidel, nyní je vysvětlena druhá část týkající se následné kontroly připojeného uživatele a přidělení pravomocí.

Zmíněný zdrojový kód obsahuje funkci *check_records*, která zkontroluje, zda uživatel má záznam v databázi. Pokud má je zavolána funkce *aware_of_records*, která zajistí zaslání informace všem moderátorům příslušné komunity o existenci záznamů nového člena.

Kód 23 vrací pravdivostní hodnotu, zda uživatel má záznam v databázi, záznam má v databázi pouze pokud je u něho evidován alespoň jeden z prohřešků.

Zdrojový kód funkce 24 se postará o zjištění a nalezení definovaných moderátorů komunity a jejich následné upozornění a sdělení o celkovém počtu evidovaných prohřešků uživatele, odkáže je jiný příkaz pomocí kterého si mohou detailněji prohlédnout o jaké prohřešky se jedná.

```

1  async def check_records(member:Member)->bool:
2      # ... gets records of member from database
3      if records:
4          return True
5      return False

```

Zdrojový kód 23: Zjištění existence záznamu uživatele

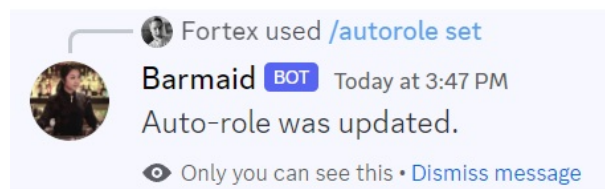
```

1  async def aware_of_records(member:Member, guild:Guild):
2      # ... gets corresponding moderators
3      # and all records of the member from database
4
5      for m in mods:
6          await m.send(f"{member} joined {guild.name} with `{len(
7              bad_records)` " +
8              "bad records.\nUse `/records @mention` on your server " +
9              "to see more information.")

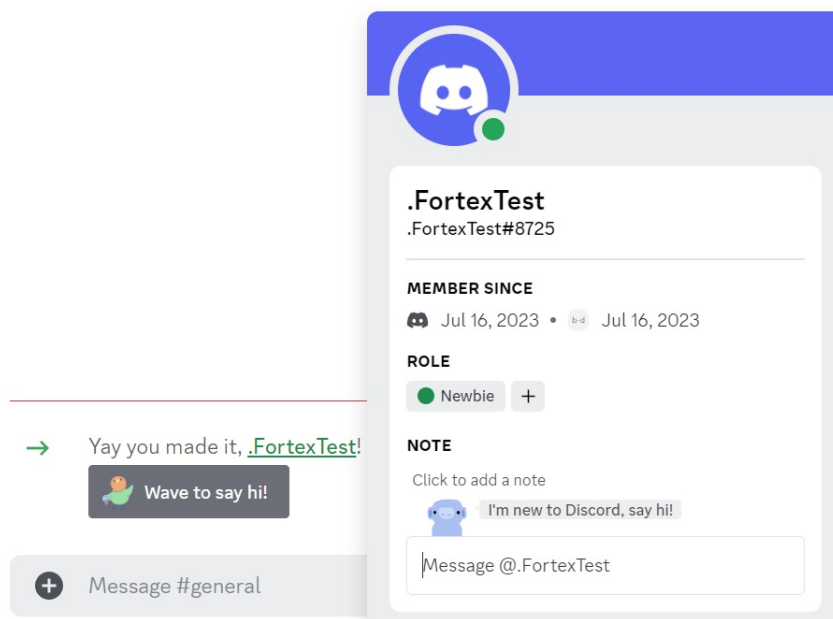
```

Zdrojový kód 24: Oznámení o existenci záznamů nového uživatele

Zdrojový kód 14 obsahuje ještě třetí funkcionalitu, přidělení role s pravomocemi novému členu, o které se nestará vlastní definovaná funkce, ale pouze nalezení definované role pro přidělení v komunitě a přidělením skrze metodu `add_roles()` třídy `discord.Member` z `discord.py` knihovny. Souvisejícími příkazy pro definování takové role k přidělení jsou: `autorole show`, `autorole set`, `autorole remove`.



Obrázek 14: Ukázka nastavení automatické role

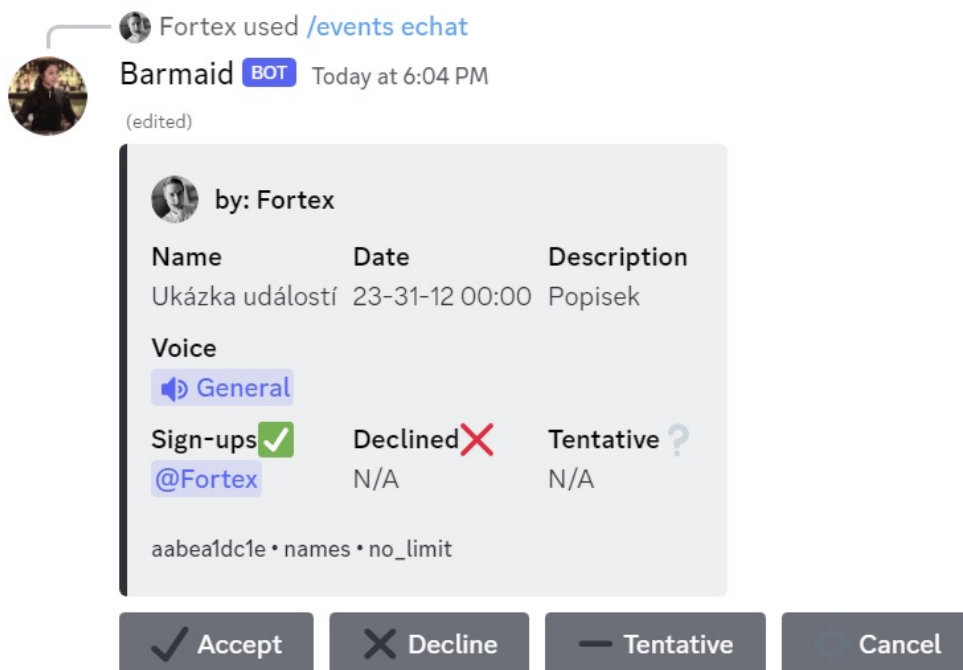


Obrázek 15: Ukázka automatického přidělení role

5.9 Události a jejich spravování

Naplánování události je zajištěno pomocí příkazu `events echat`, zkratkou za „events external chat“, tento příkaz vytvoří událost s definovaným jménem, popisem, místem a datem konání a možnostmi uvádět celá jména uživatelů při jejich evidenci nebo limit počtu uživatelů, kteří se mohou přihlásit. Zároveň s tím, pokud je definované datum ve správném formátu, je automaticky vytvořeno oznámení 15 minut před počátkem události. Nerozpoznání datumu a času dává autorovi možnost mít zde libovolný text bez funkcionality oznámení předem. Příkaz je definovaný v modulu `event.py` ve složce `events`, která zároveň definuje pohled `EventView`, kterým lze přidat tlačítka s určitou logikou ke zprávě a zaslat je společně.

Vytvořenou událost lze vidět na obr. 16, která je jako zapouzdřená zpráva formátovaná do sloupců a řádků pro přehlednost. Pod samotnou zprávou jsou také tlačítka, které k ní přísluší a intereagují. Tato tlačítka doplňují uživatele, který jej zmáčkl do příslušného seznamu v události, tlačítko „Cancel“ slouží ke zrušení události a může ji využít pouze autor.



Obrázek 16: Ukázka vytvořené události příkazem `events echat`

Na zdrojovém kódu 25 je vidět mezi řádky č. 11 až 17, které definují zapouzdřenou zprávu a formátují její položky (datum, místo, název apod.) do sloupců a řádků, řádek č. 19 se stará o uložení do databáze. Nejpodstatnější je řádek č. 21, který odesílá formátovou událost zapouzdřenou ve zprávě jako objekt třídy `discord.Embed` a vlastními definovanými tlačítky třídou `EventView`. Na řádku č. 23 je obsluhuje a nastavení oznámení o události, je-li příkaz vhodně spuštěn s argumentem `start_time`, který respektuje určitý formát data a času.

Funkce `setup_notification` ze zdrojového kódu 26 je rozdělena do dvou částí, první část se pokusí rozpoznat uživatelem stanovený čas, pokud se jedná o validní formát a aktuální čas je déle než 15 minut před zahájením události, pozastaví se vykonávání této funkce o příslušný čas. Po jejím opětovném probuzení si zkontroluje obsah zprávy bota obsahující událost, pokud existuje a nebyla zrušena, informuje do příslušného textového kanálu o tom, že se událost blíží, rovněž vezme všechny uživatele jenž se mezitím na událost přihlásili a upozorní je v soukromém chatu.

```
1 @events.command()
2 async def echat(ctx: commands.Context, include_names: bool,
3               title: str, description: str,
4               start_time: str, voice: VoiceChannel,
5               limit: int=0):
6     # ... head of the function here
7
8     # create buttons to message
9     buttons = EventView()
10
11     # embedded message for the event
12     emb = Embed(color=int("0x2f3136", 0))
13     emb.set_author(name=f"by: {ctx.author.display_name}", icon_url=
14                   ctx.author.avatar.url)
15     emb.add_field(name="Name", value=title, inline=True)
16     emb.add_field(name="Date", value=original_time, inline=True)
17     emb.add_field(name="Description", value=description, inline=
18                   True)
19     # ... rest of definition of all fields
20
21     # ... saves the event into database
22
23     event_message = await ctx.send(embed=emb, view=buttons)
24     try:
25         await setup_notification(ctx, emb, event_message.id,
26                                 start_time)
27     except ValueError:
28         # ... error message about incorrect datetime for optional
29         # notifications
30         await ctx.send(content=error_message,
31                         delete_after=S.DELETE_COMMAND_ERROR, ephemeral=True)
```

Zdrojový kód 25: Část funkce pro vytvoření události

```

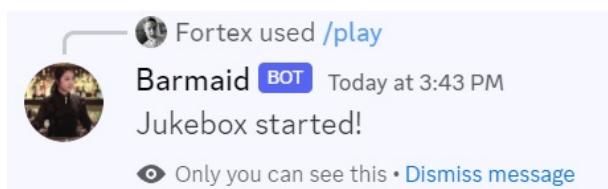
1  async def setup_notification(ctx:Context, emb:Embed, message_id:int,
    time:str):
2      # ... try parse time into format, subtract 15 minutes
3      # that time will be used to sleep the function
4
5      # wait the desired time
6      await asyncio.sleep(to_wait.seconds)
7
8      # only notify if the event wasnt cancelled
9      new_updated_message = await ctx.fetch_message(message_id)
10     new_updated_embed = new_updated_message.embeds[0]
11     event_name = new_updated_embed.fields[0]
12     if not "Cancelled: " in event_name.value:
13         # Notify in the channel
14         await ctx.send(f"@here Event `{event_name.value}` starting
            soon!",
15                         delete_after=S.DELETE_COMMAND_INVOKE)
16
17         # Notify the signed up members
18         # ... get the signed users
19         for u in user_mentions:
20             await u.send(f"Event `{event_name.value}` on `{ctx.guild.
                name}` is starting soon!")

```

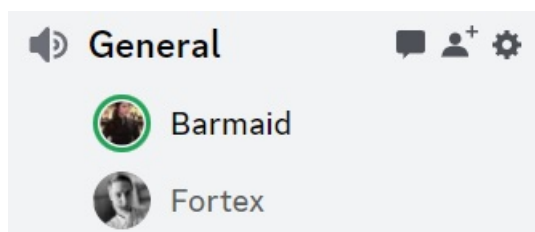
Zdrojový kód 26: Část funkce o oznámení události

5.10 Přehrávač

O přehrávání audio stop se stará modul *music.py*, který zajišťuje kontrolu nad audiem a samotné příkazy týkající se přehrávání audia. Modul obsahuje příkazy *stop*, *pause*, *resume* pro zastavení audia a odpojení bota z hlasového kanálu, pozastavení přehrávané skladby a opětovném pokračování v přehrávání. Příkazy *next*, *volume*, *queue*, *shazam* slouží k přeskočení hudby, nastavení hlasitosti bota, zobrazení fronty audio obsahu v pořadí a zjištění názvu audia. Pro zpracování playlistu slouží funkce *play_playlist*, která dá do fronty každý přístupný obsah z playlistu. Příkaz *play* přidá pouze jedno audio. Pokud se stane, že je některé audio právě přehrávané, tak bude zařazeno do stejné fronty jako používá příkaz *play_playlist*.



Obrázek 17: Ukázka zaregistrování audia



Obrázek 18: Ukázka přehrávání audia botem

Zdrojový kód [27](#) ukazuje na řádcích č. 4-6 připojení bota do kanálu, kde se nachází autor příkazu, řádek č. 8 obsluhu přehrávací fronty, na řádku č. 10 dochází k získání informací o audio potřebné například pro seznam fronty, řádky č. 13-19 se postarají o spuštění audia pomocí funkce *play_next*.

```
1 @commands.hybrid_command(with_app_command=True)
2 @commands.guild_only()
3 async def play(ctx: commands.Context, url: str):
4     # ... get already connected client
5     voice_client = await ctx.author.voice.channel
6                     .connect(timeout=30, reconnect=True)
7
8     # ... gets queue and puts the url into it
9
10    # ... extracts the info about audio and processes it
11
12    # start playing the next song if not already playing
13    if not voice_client.is_playing():
14        await play_next(ctx)
15        await ctx.send("Jukebox started!",
16                        delete_after=S.DELETE_EPHEMERAL)
17    return
18    await ctx.send("Added to queue!",
19                  delete_after=S.DELETE_EPHEMERAL)
```

Zdrojový kód 27: Přehrávání audia příkazem *play*

Funkce ze zdrojového kódu 28 zprostředkovává informace o aktuálně přehrávané hudbě na řádku č. 2, na řádku č. 3 dochází k ukončení přehrávání je-li fronta vyčerpána, jinak se z fronty získává odkaz audia (z platformy YouTube), který je následně zpracován na řádcích 5-9. Před odesláním k přehrávání se provádí transformace audia do příslušné hlasitosti a formátu na řádku č. 10. O samotné přehrávání audia se poté starají řádky č. 12-14, které audio spustí v příslušném kanále komunitního serveru a nastaví se po jeho ukončení opět na zavolání této funkce, kde právě přehrané audio už bylo odstraněno z fronty.

```
1  async def play_next(ctx:commands.Context):
2      # ... gets the url from queue
3      # or stops playing when queue is empty
4
5      loop = asyncio.get_event_loop()
6      data = await loop.run_in_executor(None, lambda: _ytdl.
7          extract_info(url, download=False))
8      music = data["url"]
9
10     player = discord.FFmpegPCMAudio(music, **_ffmpeg_options)
11     # ... transforms the audio
12
13     # ... gets the voice client
14     voice_client.play(player, after=lambda e:
15         asyncio.run_coroutine_threadsafe(play_next(ctx), loop=loop))
16     # ... adds additional information to voice client
17     # for example currently playing audio
```

Zdrojový kód 28: Zpracování a přehrávání audia

5.11 Hosting aplikace bota

Pro nasazení aplikace je zvolena hostingová služba nazývajícím se Amazon Cloud Web Services [11] se zvoleným hardwarem pronajatého stroje s jedním virtuálním procesorem a 1GB operační pamětí běžící na operačním systému Ubuntu 22.04 LTS, architektura 64-bit (x86). Tento způsob je zvolen na základě dostačujícího výkonu pro aplikaci bota a použitím bezplatného účtu pro takto výpočetně nenáročný stroj. Se vzdáleným přístupem k tomuto stroji došlo k přesunutí celé aplikace bota na tento stroj, kde jsou doinstalovány všechny potřebné prerekvizity podle přiloženého souboru *readme.txt* u této aplikace, následně aplikace byla spuštěna.

Závěr

Cílem práce bylo vytvořit aplikaci bota pro platformu Discord, která umožní zlepšení uživatelské zkušenosti na platformě správou komunity, správou uživatelů, plánováním událostí a zprostředkováním audio přehráváče.

Teoretická část práce obsahuje seznámení čtenáře s platformou Discord a její funkcionalitou. Popisuje technologie, jež platforma nabízí, jak fungují a jak se s nimi zachází. Popisuje také technologie, se kterými aplikace bota pracuje navíc od samotných technologií platformy.

V praktické části je popsán návrh aplikace bota, který by měl umět ke splnění zadané požadavky a detailněji popisuje jejich vlastnosti. Poté popisuje způsoby, jakými byl návrh implementován a použit.

Výstupem práce je bot, který umožňuje moderátorům komunity na platformě Discord spravovat jejich komunitu způsoby jako je přiřazení pravomocí pomocí vlastní definované role, kontrolou nově připojených uživatelů a jejich záznamů trestů, které má bot dostupné a sám si je vytvořil. Umožňuje také stanovit pravidla platná pro komunitu a tyto pravidla automaticky zaslat nově příchozímu uživateli. Bot umožňuje uskutečnit události, které jsou naplánovány pomocí příkazu a jsou zobrazeny a evidovány přímo v chatu. Uživatelé se na tuto událost mohou přihlásit a budou moci být upozorněni před počátkem konání. Aplikace umožňuje také zlepšení uživatelské zkušenosti audio-přehrávačem ve hlasovém hovoru v kanálech komunity.

Conclusions

Goal of this thesis was to create an application for Discord platform, which allows its users to enhance their experience on the platform with better management of the guild communities and its members, mediate an events organizer and audio player.

Theoretical part of the thesis contains an introduction to the Discord platform and its functionality. Description of technologies which the platform offers, how they work and how to use them. It also describes technologies which the application uses in addition to the platform itself.

The practical part describes the design of the application, which should be able to meet the requirements and more detailed description of its features, then there are described the ways in which the design was implemented.

The output of the thesis is a bot, which allows moderators of the communities on the Discord platform to manage their community in ways such as assigning privileges using their own defined role. Checking newly connected users and their punishments records, which the bot has available and created for himself before. Allowing to create own set of rules for the community and automatically send them out to new users. The bot allows to create and hold for events, which are planned using a command and are displayed directly in the chat, users can register for this event and can be notified before the start of the event. The application also allows for an audio player in a voice channel in the guild community to improve the user experience.

A Obsah příloženého datového média

theses/

Kompletní text této práce v PDF formátu.

src/

Kompletní zdrojové soubory implementované aplikace, jejími konfiguračními soubory a souborem obsahující všechny knihovny třetích stran pro jejich instalaci skrze program `pip`, dostupný po nainstalování interpretu programovacího jazyka popsáném v souboru s instrukcemi `readme.txt`.

readme.txt

Instrukce pro instalaci a spuštění programu *barmaid.py*, včetně všech požadavků pro jeho bezproblémový provoz. / Instrukce a webová adresa, na které je aplikace nasazena pro účel testování při tvorbě posudků práce a pro účel obhajoby práce.

Seznam zkratek

API Application Programming Interface

CLI Command Line Interface

HTTP Hypertext Transfer Protocol

JSON JavaScript Object Notation

ToS Terms of Service

UI User Interface

URL Uniform Resource Locator

VoIP Voice over Internet Protocol

Literatura

- [1] Wikipedia. *Final Fantasy XIV*. [online]. 2013. [cit. 2023-16-03]. Dostupné z: https://en.wikipedia.org/wiki/Final_Fantasy_XIV
- [2] Wikipedia. *Discord*. [online]. 2014. [cit. 2023-16-03]. Dostupné z: <https://en.wikipedia.org/wiki/Discord>
- [3] Chanty. *Slack vs Discord – Gaming, Working or Both?* [online]. 2022. [cit. 2023-16-03]. Dostupné z: <https://www.chanty.com/blog/discord-vs-slack/>
- [4] Discord. *Rate Limits*. [online]. 2015. [cit. 2023-17-03]. Dostupné z: <https://discord.com/developers/docs/topics/rate-limits>
- [5] Discord. *Discord API*. [online]. 2015. [cit. 2023-17-03]. Dostupné z: <https://discord.com/developers/docs/reference#api-reference>
- [6] Discord. *Gateway*. [online]. 2015. [cit. 2023-17-03]. Dostupné z: <https://discord.com/developers/docs/topics/gateway>
- [7] Toptal. *How to Make a Discord Bot: an Overview and Tutorial*. [online]. 2010-2023. [cit. 2023-17-03]. Dostupné z: <https://www.toptal.com/chatbot/how-to-make-a-discord-bot>
- [8] Caleb Hattingh. *Using Asyncio in Python*. O'Reilly Media, Incorporated, 2020. ISBN 978-149-2075-332
- [9] Github. *discord.py*. [online]. 2023. [cit. 2023-18-03]. Dostupné z: <https://github.com/Rapptz/discord.py>
- [10] Raspberry. *Computing for everybody*. [online]. 2012. [cit. 2023-18-03]. Dostupné z: <https://www.raspberrypi.com/>
- [11] Amazon. *What is cloud computing?* [online]. 2006. [cit. 2023-18-03]. Dostupné z: <https://aws.amazon.com/what-is-cloud-computing/>
- [12] Discord. *Community Resources*. [online]. 2015. [cit. 2023-18-03]. Dostupné z: <https://discord.com/developers/docs/topics/community-resources#libraries>
- [13] Interactive Bees Blog. *Asyncio - vylepšené asynchronní paradigma*. [online]. 2022. [cit. 2023-18-03]. Dostupné z: <https://blog.neonkid.xyz/283>
- [14] Wikipedia. *FFmpeg*. [online]. 2007. [cit. 2023-18-03]. Dostupné z: <https://cs.wikipedia.org/wiki/FFmpeg>
- [15] FFmpeg. *Ffmpeg*. [online]. 2023. [cit. 2023-18-03]. Dostupné z: <https://ffmpeg.org/>